D2.3 – System Architecture specification (a)

H2020 ICT-04-2015

Disaggregated Recursive Datacentre-in-a-Box
Grant Number 687632

D2.3 – System Architecture specification (a)

WP2: Requirements and Architecture Specification, Simulations and Interfaces

<table>
<thead>
<tr>
<th>Due date:</th>
<th>PM6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submission date:</td>
<td>30/06/2016</td>
</tr>
<tr>
<td>Project start date:</td>
<td>01/01/2016</td>
</tr>
<tr>
<td>Project duration:</td>
<td>36 months</td>
</tr>
<tr>
<td>Deliverable lead organization</td>
<td>BSC</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td><strong>Version:</strong></td>
<td>1.2</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Final</td>
</tr>
</tbody>
</table>
| **Author(s):**               | Maciej Bielski (VOSYS)  
Christian Pinto (VOSYS)  
Dimitris Syrivelis (UTH)  
Dionisios Pnevmatikatos (FORTH)  
Dimitris Theodoropoulos (FORTH)  
Theoni Alexoudi (UTH)  
Andrea Reale (IBM)  
Yiannis Gkoufas (IBM)  
Ferad Zyulkyarov (BSC)  
Jose Carlos Sancho (BSC)  
Mario Nemirovsky (BSC)  
Hugo Meyer (BSC)  
Josue Quiroga (BSC)  
George Zervas (UoB), Vaibhawa Mishra (UoB)  
Adirinjo Peters (UoB)  
Hui Yaan (UoB)  
Evert Pap (SINTECS) |
| **Reviewer(s):**             | Evert Pap (SINTECS)  
Kostas Katrinis (IBM) |
| **Dissemination level**      | PU Public |

**Disclaimer**

This deliverable has been prepared by the responsible Work Package of the Project in accordance with the Consortium Agreement and the Grant Agreement No 687632. It solely reflects the opinion of the parties to such agreements on a collective basis in the context of the Project and to the extent foreseen in such agreements.
Acknowledgements

The work presented in this document has been conducted in the context of the EU Horizon 2020. dReDBox (Grant No. 687632) is a 36-month project that started on January 1st, 2016 and is funded by the European Commission.

The partners in the project are IBM IRELAND LIMITED (IBM-IE), PANEPISTIMIO THESSALIAS (UTH), UNIVERSITY OF BRISTOL (UOB), BARCELONA SUPERCOMPUTING CENTER – CENTRO NACIONAL DE SUPERCOMPUTACION (BSC), SINTECS B.V. (SINTECS), FOUNDATION FOR RESEARCH AND TECHNOLOGY HELLAS (FORTH), TELEFONICA INVESTIGACION Y DESSARROLLO S.A.U. (TID), KINESENSE LIMITED (KS), NAUDIT HIGH PERFORMANCE COMPUTING AND NETWORKING SL (NAUDIT HPC), VIRTUAL OPEN SYSTEMS SAS (VOSYS), POLATIS LTD. (POLATIS).

The content of this document is the result of extensive discussions and decisions within the dReDBox Consortium as a whole.

More information

Public dReDBox reports and other information pertaining to the project will be continuously made available through the dReDBox public Web site under http://www.dReDBox.eu.
## Version History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Comments, Changes, Status</th>
<th>Authors, contributors, reviewers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>24/05/16</td>
<td>First draft</td>
<td>D. Theodoropoulos (FORTH)</td>
</tr>
<tr>
<td>0.2</td>
<td>08/06/16</td>
<td>System software, virtualization and orchestration tools</td>
<td>M. Bielski (VOSYS), C. Pinto (VOSYS), D. Syrivelis (UTH), A. Reale (IBM), Yiannis Gkoufas (IBM)</td>
</tr>
<tr>
<td>0.5</td>
<td>14/06/19</td>
<td>UoB text integration</td>
<td>D. Pnevmatikatos, (FORTH), D. Theodoropoulos (FORTH)</td>
</tr>
<tr>
<td>0.6</td>
<td>16/06/16</td>
<td>Added RMST description, related work in the introduction, the appendix, and a first draft of Section 6</td>
<td>D. Pnevmatikatos, (FORTH), D. Theodoropoulos (FORTH)</td>
</tr>
<tr>
<td>0.7</td>
<td>22/06/2016</td>
<td>Removed figures for Rack scale architecture with Heterogeneous trays with pure optical network interconnect. Added Section 3.4 for cable complexity. Include Table 1 for comparison between optical switch and Electrical connectivity</td>
<td>Vaibhawa Mishra (UoB), Adiranjo Peters (UoB), Hui Yaan (UoB)</td>
</tr>
<tr>
<td>0.8</td>
<td>22/06/2016</td>
<td>Changed Figure 8 tray architecture design and updated the description of tray architecture in Section 2.2.</td>
<td>Hugo Meyer (BSC), Jose Carlos Sancho (BSC), Josue Quiroga (BSC), Ferad Zyulkyarov (BSC), Mario Nemirovsky (BSC)</td>
</tr>
<tr>
<td>0.9</td>
<td>27/06/2016</td>
<td>Written sections executive summary and conclusions. Integrated changes. Removed figure 9.</td>
<td>Jose Carlos Sancho (BSC), Ferad Zyulkyarov (BSC)</td>
</tr>
<tr>
<td>1.0</td>
<td>30/06/2016</td>
<td>Final Editing and Review</td>
<td>Kostas Katrinis (IBM)</td>
</tr>
<tr>
<td>1.1</td>
<td>26/10/2016</td>
<td>Restructuring and Revision</td>
<td>Kostas Katrinis (IBM), Hugo Meyer (BSC), Jose Carlos Sancho (BSC), Josue Quiroga (BSC), Ferad Zyulkyarov (BSC)</td>
</tr>
<tr>
<td>1.2</td>
<td>08/11/2016</td>
<td>Final editing</td>
<td>Josue Quiroga (BSC), Ferad Zyulkyarov (BSC), George Zervas (UoB)</td>
</tr>
</tbody>
</table>
Table of Contents

Executive Summary .................................................................................................... 6
List of Acronyms and Naming Conventions ................................................................ 7
1 Introduction ........................................................................................................ 10
2 System Architecture ........................................................................................... 13
   2.1 Overview and Naming Conventions ........................................................... 13
   2.2 dBRICKs Architecture ................................................................................. 16
      2.2.1 Compute Brick Architecture (dCOMPUBRICK) ............................... 16
      2.2.2 Memory Brick Architecture (dMEMBRICK) ...................................... 19
      2.2.3 Acceleration Brick Architecture (dACCELBRICK) .......................... 20
   2.3 Tray Architecture (dTRAY) ......................................................................... 21
      2.3.1 Power Consumption Estimation .......................................................... 23
   2.4 Rack Architecture (dRACK) ........................................................................ 23
3 dReDBox Hierarchical Network Architecture ..................................................... 24
   3.1 System-level operation ............................................................................... 24
   3.2 Intra-tray connectivity (within dTRAY) ........................................................ 26
   3.3 Inter-tray connectivity (between dBOXes) .................................................. 26
   3.4 Cabling Complexity ..................................................................................... 27
4 Scalability Analysis and Considerations ............................................................ 28
   4.1 dBRICK scalability ...................................................................................... 28
   4.2 dTRAY scalability ........................................................................................ 28
   4.3 dBOX scalability ........................................................................................ 29
   4.4 dRACK scalability ....................................................................................... 30
   4.5 Datacenter scalability .................................................................................. 30
5 Orchestration tools and System Software Architecture ..................................... 31
   5.1 Rack-scale software defined memory system architecture ........................ 32
   5.2 Memory allocation and deallocation processes .......................................... 32
   5.3 Orchestration tools high-level architecture ................................................. 33
      5.3.1 Orchestration tools .............................................................................. 34
      5.3.2 Compute brick system software .......................................................... 35
      5.3.3 Virtual machine management .............................................................. 36
   5.4 Shared memory and device virtualization ................................................... 38
6 Implementation considerations .......................................................................... 40
7 Conclusions ....................................................................................................... 41
8 References ........................................................................................................ 42
9 Appendix I – SoC Selection Process .................................................................. 43
10 Appendix II - Arrayed Waveguide Grating Router (AWGR) based solutions .... 46
Executive Summary

This document describes the initial architecture design and specification of the prototype dReDBox system. The document was prepared as a result of a series of technical discussions among the project partners and their direct input. The design and specification described in this document are driven by the requirements set in deliverable D2.1.

Unlike traditional datacenters where the basic building block is a monolithic computer node, the basic building resources in the dReDBox system are called bricks. Each brick will be a standard hot-pluggable module. The dReDBox system will be composed of three types of bricks:

1. Compute brick – module with SoC that will run virtual machines
2. Memory brick – module with pool of memory to be used as system memory by the compute bricks
3. Accelerator brick – module with programmable logic that will speed up kernel computation

The proposed dReDBox architecture is a disaggregated rack-scale system that consists of arbitrary types of bricks. The system is anticipated to be organized into standard 2U rack tray units which will be mounted to a rack. System resources are aimed to be connected via multiple networks. High-speed and low-latency electrical network will be used for intra-tray data access in memory bricks. High-speed low-latency optical network will be used for inter-tray data access in memory bricks. PCIe network will be used for signaling and remote interrupts. Ethernet network will be used for data communication, provisioning and platform management.

System software and orchestration tools will be used to provision, access and manage the system resources. Specifically, the system software will implement software-defined memory where the memory is accessed via network. The orchestration software and tools will manage the allocation and deallocation of memory, sharing of memory address space between virtual machines and also access the peripheral devices as well as accelerators.
List of Acronyms and Naming Conventions

Processor Core or Compute Core or Core or Processing Unit (PU)
An independent processing unit that reads and execute machine program instructions. Manufacturers typically integrate multiple cores onto a single integrated circuit die (known as a chip multiprocessor or CMP), or onto multiple dies in a single chip package.

Multi-core processor
A multi-core processor implements multiprocessing in a single physical package. Designers may couple cores in a multi-core device tightly or loosely. For example, cores may or may not share caches, and they may implement message passing or shared-memory inter-core communication methods.

LLC
Last Level Cache. A CPU cache is a hardware cache used by the central processing unit (CPU) of a computer to reduce the average cost (time or energy) to access data from the main memory. Most CPUs have different independent caches, including instruction and data caches, where the data cache is usually organized as a hierarchy of more cache levels (L1, L2, etc.). The shared highest-level cache, which is called before accessing memory, is usually referred to as the Last Level Cache (LLC).

Memory Controller (MC)
Memory controllers contain the logic necessary to read and write to DRAM, and to "refresh" the DRAM. Without constant refreshes, DRAM will lose the data written to it as the capacitors leak their charge within a fraction of a second (not less than 64 milliseconds according to JEDEC standards).

Hypervisor
A hypervisor, or virtual machine monitor (VMM), is a piece of computer software, firmware or hardware that creates and runs virtual machines.

IaaS
Infrastructure as a Service (IaaS) is a form of cloud computing that provides virtualized computing resources over the Internet. IaaS is one of three main categories of cloud computing services, alongside Software as a Service (SaaS) and Platform as a Service (PaaS). In an IaaS model, a third-party provider hosts hardware, software, servers, storage and other infrastructure components on behalf of its users. IaaS providers also host users' applications and handle tasks including system maintenance, backup and resiliency planning.

KVM
Kernel-based Virtual Machine (KVM) is a virtualization infrastructure for the Linux kernel that turns it into a hypervisor. KVM requires a processor with hardware virtualization extensions.

libvirt, libvirtd
A toolkit to interact with the virtualization capabilities of recent versions of Linux (and other OSes). libvirt provides all APIs needed to do the management, such as: provision, create, modify, monitor, control, migrate and stop virtual domains - within the limits of the support of the hypervisor for those operations. The
### Direct Memory Access (DMA)

Direct memory access (DMA) is a feature of computer systems that allows certain hardware subsystems to access main system memory (RAM) independently of the central processing unit (CPU).

### NUMA

NUMA (non-uniform memory access) is the phenomenon that memory at various points in the address space of a processor have different performance characteristics. These systems required modified operating-system kernels with NUMA support that explicitly understood the topological properties of the system's memory (such as the chassis in which a region of memory was located) in order to avoid excessively long signal path lengths. The designers of these products had to modify the Linux kernel to support NUMA.

### Openstack

OpenStack software controls large pools of compute, storage, and networking resources throughout a datacenter, managed through a dashboard or via the OpenStack API. OpenStack works with popular enterprise and open source technologies making it ideal for heterogeneous infrastructure.

### Nova

Nova is an OpenStack project designed to provide power massively scalable, on demand, self-service access to compute resources.

### OS

Operating System

### QEMU

QEMU is a generic and open source machine emulator and virtualizer. When used as a machine emulator, QEMU can run OSes and programs made for one machine (e.g. an ARM board) on a different machine (e.g. your own PC). When used as a virtualizer, QEMU achieves near native performance by executing the guest code directly on the host CPU.

### SDM Agent

The Software Defined Memory daemon agent running on dReDBox compute bricks to facilitate remote provisioning, modification, control and monitoring of virtual machines.

### SDM Controller

The Software Defined Memory Controller is a centralized control entity orchestrating resource management and allocation and power management across disaggregated of a dReDBox datacenter.

### VM

Virtual Machine – Isolated virtualization unit running its own guest operating systems

### VMM

Virtual Machine Monitor. *See Hypervisor*

### Mid-Board Optics (MBO)

MBO is the natural transition technology from current front-panel transceivers to more integrated electronic-optical devices. It avoids the front-panel bottleneck, improves ports and bandwidth.
scaling of the rack space and may help to solve the packaging bottleneck.
1 Introduction

The architecture of servers has always been designed around trays with a fixed number of different, tightly coupled with each other components such as processor, memory, and peripherals. Focusing here on datacenters used to materialize e.g. cloud or hyperscale computing, conventional architectures thereof typically house a large set of interconnected racks, each utilizing multiple interconnected server trays, as abstractly illustrated in Figure 1. In general, a server tray consists of – typically multiple – processing unit(s) (PU) attached via one (resp. multiple) Memory Controller(s) (MC) to tray-local Random Access Memory (RAM) for rapid instruction read and fast, random read/write byte-level access to data. The PU can also access persistent local storage and I/O devices (e.g. flash storage, accelerators) using a single or a hierarchy of I/O bridges. Similarly, each server tray also hosts one or multiple Network Interface Cards (NICs); the latter leverage physical connectivity to dedicated network switches and routers (and rarely also switching capability that is available on-board on multi-port NICs) to facilitate data exchange between PUs that reside in distinct server trays and potentially in distinct racks.

Figure 1 – Contemporary datacenter architecture; each rack comprises a set of server trays and multiple racks are interconnected to form scalable processing platforms.

This approach introduces limitations in terms of available system resources and scalability, leading to spare resource fragmentation and inefficiencies in current datacenters.

At the same time, current datacenter scale-out workloads mostly perform parallel tasks (e.g. internet search) that require access to vast amounts of data. Hence, to tackle the architecture’s poor scalability, previous related works have already suggested various approaches, primarily targeting the reduction of remote memory access latency.

For example, in [6] the authors show that such workloads mostly require communication between compute cores and LLCs, i.e. there is minimal core-to-core communication and data coherence activity. Based on these observations, they propose a scale-out processor architecture, called NoC-Out that decouples LLCs from the cores, placing them in the die center, which eliminates direct core-to-core communication, thus reducing the on-chip network complexity. When evaluated on a 64-core CPM, NoC-Out delivers performance improved by 17%, while reducing network area by 28% compared to a tiled mesh-based design.

Furthermore, in [7] the authors propose the Scale-out NUMA (SO-NUMA) architecture that exposes remote memory at user level via a remote memory controller (RMC), which converts memory transactions (e.g. RDMAs) to network operations, hence bypassing the OS. Moreover, the RMC is tightly integrated to the processor’s cache coherence hierarchy; results suggest that remote memory access is reduced close to 4x compared to local memory transactions.
The work described in [8] presents an interesting study on how different possible network interface (NI) placements along many-core SoCs can affect the network performance. The authors compare three main approaches, (a) the NI placed at the chip’s edge, (b) each core integrating an NI, and (c) splitting memory request operations from data accesses, where the former is replicated within each core, and the latter placed at the chip boundaries. Results suggest that the latter approach delivers the high bandwidth of (a) and low latency of (b).

On the other hand, dReDBox aims to deliver a disaggregated architecture that will not require memory or accelerator to be co-located with processor in the same node. This will enhance elasticity, improved virtual machine migration, and reduce total cost of ownership than the current datacenters. In this new architecture the main building block is not anymore the server, but the brick. Different bricks are designed with different functionality like compute, memory and peripheral bricks.

To build datacenters based on disaggregated architecture requires new breakthrough developments in network, memory interface, hypervisor and orchestration layer. dReDBox will advance and address these requirements during the project duration.

Specifically, the network should provide ultra-low latency and high bandwidth to efficiently interconnect disaggregated components in the datacenter. Moreover, scalable network switches should be used to interconnect large number of components. In addition, it is desirable that the network configuration should be performed dynamically even during the application execution. And finally, low power interconnects will be employed to significantly reduce the cost of owner-ship in datacenters.

Regarding memory, interfaces should be transparent to application. Remote memory should require no changes in current applications and because of that it should be accessed as if it is local memory in today’s systems. Moreover, remote interrupt delivery support for Direct Memory Access (DMA) should be provided to interconnect with peripherals. Extensions in hypervisors’ memory management and configurable logic will coordinate to provide the transparent access to remote memory.

Finally, the hypervisor and management layer is aimed to support virtual memory ballooning and hot-pluggable operations of different bricks. A new software-defined global memory and peripheral resource management will be needed to configure on-the-fly network devices. In addition, the management layer will also reduce drastically power through resource management and efficient scheduling of processes/virtual machines. Integration with existing standard datacenter resource management will be provided through APIs.

dReDBox aims to showcase its superior value through validation across multiple user cases from relevant industries. Three use cases have been identified in the project to illustrate the advantages of this new architecture. They are video-analytics, network analytics, and network function virtualization. The project will demonstrate that these use cases will directly benefit from accessing a very fast and large shared pool of memory and peripherals in dReDBox. Multiple terabytes of memory will be available for the first time to applications. This new capability will drive application efficiencies in different commercial workloads. In general, it is envisioned that datacenter workloads using multiple terabytes of data will have a substantial benefit of the dReDBox architecture since it will significantly reduce the negative effect of memory trashing due to the limited amount of available memory to processes in today’s servers. In contrast, dReDBox will provide vast amounts of memory reducing or even eliminating the need to memory swapping to disk.

This deliverable includes the following sections:

- Section 2 – System Architecture. In this section the architecture of the system will be described including the architecture of the brick, the tray and the whole...
D2.3 – System Architecture specification (a)

- Section 3 – Network Architecture. It describes the different multi-component connectivity and the corresponding network hierarchies and components.
- Section 4 – Explores scalability across dRedBox architecture components and discusses relevant considerations.
- Section 5 – Orchestration tools and System Software Architecture. This section describes the design of the compute brick operating system/hypervisor and of orchestration software.
- Section 6 – Implementation considerations discusses various potential issues related to the implementation and their candidate solutions.
2 System Architecture

This section presents the overall architecture of a dReDBox datacenter. The presentation starts with an overview of a dReDBox 2U carrier box¹, followed by an overview of a dReDBox rack; a set of multiple dReDBox racks builds up – interconnected via an appropriate datacenter network - to form a dReDBox datacenter. As part of the overview, we also include a comprehensive definition of components and their naming conventions specific to dReDBox. Subsequently, we present the rack-level architecture, following a bottom-up approach. While inevitably some detail is given pertaining to the memory/data interconnect, we defer a comprehensive presentation thereof until Section 3.

2.1 Overview and Naming Conventions

The dReDBoX architecture comprises pluggable compute/memory/accelerator modules (termed “bricks” in dReDBox terminology) as the minimum field replaceable units. A single or sets of multiples of each brick type forms an IT resource pool of the respective type. A mainboard tray with compatible brick slots and on-board interconnection, flash storage and baseboard management components is used to carry (up to 16) bricks. A 2U carrier box (visually corresponding from the outside to a conventional, rack-mountable datacenter server) in turns hosts the mainboard tray and the intra-tray optical switch modules.

Figure 2- Overview of a dReDBox carrier box (“dReDBox-Box”), hosting a tray that carries arbitrary combinations of compute/memory/accelerator bricks, an electronic switch matrix for intra-tray connectivity and a set of optical switches for off-tray interconnection.

¹ We deliberately avoid to use the term “dReDBox server” to distinguish from the ramifications that conventional datacenter server structures bring to utilization, Total Cost of Ownership and further limitations, as discussed in details in Section 1.
Figure 2 illustrates the design briefly described above. The figure is best understood by defining the nomenclature and the concise scope and role of each of its main and distinguishing constituent components, as given below:

**dBOX**
A dReDBox-Box houses the main components of the dReDBox system and can be considered the heart of the dReDBox system. The dBOX will be compatible with standard datacenter infrastructures and will look like any other server.

**dTRAY**
A dReDBox-Tray provides the interconnection and supporting functions for the different dReDBox-modules. It serves as a “motherboard” in the dBOX.

**dBRICK**
A dReDBox-Brick forms the minimum, independently replaceable unit in the dReDBox datacenter. There are three different types of dReDBox-Bricks: compute, memory and accelerator bricks. At any hierarchy level, dBRICKs are interchangeable and can be deployed in arbitrary combinations to closely match service provider and/or user needs.

**dCOMPUBRICK**
The dReDBox-Compute-Brick constitutes the minimum replaceable unit providing general-purpose application processing to the dReDBox datacenter.

**dMEMBRICK**
The dReDBox-Memory-Brick constitutes the minimum replaceable unit providing disaggregated memory to the dReDBox datacenter.

**dACCELBRICK**
The dReDBox-Accelerator-Brick constitutes the minimum replaceable unit providing programmable, application-specific accelerated processing to the dReDBox datacenter. It will also have the ability to interface with a 100GbE interface on the dTRAY.

**dBESM**
The dReDBox-Box-ESM is a Custom-Off-The-Shelf (COTS) Electrical Switch Matrix (ESM) used to interconnect dBRICKs residing with the same dBOX.

**dBOSM**
The dReDBox-Box-OSM is a COTS Optical Switch Matrix (OSM) used to interconnect dBRICKs residing within a dBOX to dBRICKs residing in remote dBOXes (either in the same or in distinct racks). The OSM can also be used for intra-tray dBRICK interconnection, coupling the ESM to increase density and/or throughput of connectivity in the tray.

The dBOX forms the basic building block to construct a dReDBoxX rack (termed “dRACK”). Unlike the dBOX, the dRACK resembles more of a conventional datacenter.
rack: carrier boxes are stacked up into a 42U rack, including network switches that facilitate intra-rack communication, as well as connectivity to the rest of the datacenter racks through the datacenter-level network.

Figure 3 – Overview of a dReDBox rack (resp. datacenter) architecture comprising several dBOXes interconnected with hybrid optical and electrical switching (“dROSM”)

Figure 3 illustrates the rack (resp. datacenter) design briefly described above. Similar to the approach followed in the previous to present the seminal components comprising a dBOX, the respective list of definitions pertaining to a dReDBox rack (resp. datacenter) is given below:

**dRACK**  
A dReDBox-Rack houses multiple, interconnected dBOXes. In the scope of the project, it forms the complete dReDBox system. The dRACK is the final Hardware deliverable associated with “D5.2: Hardware integration and tests of all bricks and tray (b)”. The dRACK will be used as the platform for the different demonstrators.

**dPERTRAY**  
The dReDBox-Peripheral-Tray is a COTS product providing convenient support for attaching different kind of peripherals (notably secondary storage) through a PCIe bus. This will be a “plug-and-play” solution which can be connected to a dBOX using a standard PCIe cable.
The **dReDBox-Rack-OSM** is a COTS Optical Switch Matrix used to interconnect dBRICKs residing in distinct dBOXes with the same dRACK. It also serves as a leaf-switch to route traffic emanating from (resp. terminated at) the local dRACK to a datacenter destination (resp. from a datacenter source) residing off the local dRACK. In the project, we also aim to experiment with an embodiment of a dROSM featuring hybrid optical/electronic (i.e. both fiber- and packet-switching).

The **dReDBox-Datacenter-OSM** is used to interconnect the different dRACKs in a datacenter. It will connect to the different dROSMs in the datacenter. The dDOSM is here referenced for the sake of completeness and to facilitate a discussion of the overall scalability potential of a dReDBox datacenters. However, its further materialization is out of the scope of the project.

A **dReDBox-Cluster** is a logical grouping of dBOXes residing within the same dRACK. The decision of sub-dividing a dRACK into dCLUSTs is mainly motivated by the port density limits of a dROSM, as the largest commercially-available dROSM is not capable of interconnecting all the dBOXes within a 42U dRACK.

The **dReDBox-Box-PSU** is an AC/DC power supply, capable of providing enough power to a fully provisioned dCLUST.

We note that the above defined nomenclature will be used throughout the project (also in the rest of this deliverable) to refer to any of the above components, forming the master section of reference for any dReDBox-specific term. While future deliverables may reiterate – partly or in their entirety – these definitions for the sake of being self-contained, or even further specialize by adding additional context-specific definitions, we will strive to keep the above set of definitions consistent and complete throughout the project.

### 2.2 dBRICKs Architecture

dBRICKs are the smallest realization unit in the dReDBox architecture. The term encompasses general-purpose processing (dCOMPUBRICK), random-access memory (dMEMBRICK) and application-specific “accelerators” (dACCELBRICK). As described above, dBRICKs will be connected to the rest of the system by means of a tray that besides connectivity will also provide the necessary power to each brick.

#### 2.2.1 Compute Brick Architecture (dCOMPUBRICK)

The dReDBox compute brick (dCOMPUBRICK) is the main processing block in the system; a block diagram depicting the architecture of the dCOMPUBRICK is shown in Figure 4. Following a rigorous comparative evaluation of candidate SoCs (cf. Appendix I – SoC Selection Process”), the consortium has decided to use the Xilinx Zynq Ultrascale+ MPSoC (EG version) [10] as the compute brick’s System-On-Chip. Among others, this choice reduces the number of components (no separate FPGA chip needed), eventually leading to smaller dBRICKs sizes and power consumption. Another benefit of using a reconfigurable SoC as PUs is their flexibility in terms of supporting access to memory modules of different technologies (e.g. HMC). The same
would require a major investment in the case of ASIC-based SoCs. Trading off these advantages, it must be noted that reconfigurable SoCs integrate less powerful software processing units, compared to multicore platforms at the same cost range.

The compute brick hosts local off-chip memory (DDR4) for low-latency and high-bandwidth instruction read and read/write data access, as well as Ethernet and PCIe ports for data and system communication and configuration. Also, each dCOMPUBRICK features QSPI-compatible flash storage (16-32 MB) and a micro-SD card socket (not shown in Figure 4) to ease flashing of firmware and boot images from locally, in case of disconnection or for debugging purposes.

![Figure 4 - Block diagram of a dCOMPUBRICK, featuring the Xilinx Zynq Ultrascale+ MPSoC (EG version) as quad-core SoC. The on-chip programmable logic on the SoC is used to host transaction glue logic, housekeeping state and communication logic required for accessing disaggregated resources.](image)

The compute brick can reach disaggregated resources, such as memory and accelerators, via dReDBox-specific glue intellectual property (termed “Transaction Glue Logic”) on the data-path and communication endpoints implemented on the programmable logic of the dCOMPUBRICK’s Zynq Ultrascale+ MPSoC. System interconnection to disaggregated resources occurs via multiple ports leading to circuit-switched tray- and rack-level interconnects (dBESM/dBOSM and dROSM respectively, as defined in Section 2.1). As also shown in Figure 4, we also plan to experiment with packet-level system/data interconnection, using Network Interface (NI) and a brick-level packet switch (also implemented on programmable logic of the dCOMPUBRICK MPSoC), on top of the inherently circuit-based interconnect substrate. While not in the critical path in terms of project delivery and impact, there is potential value in such an approach, specifically in terms of increasing the connectivity of a dCOMPUBRICK due to multi-hopping and thus creating an opportunity to increase the span of resource pools reachable from a single dCOMPUBRICK.

For each request arriving to the Transaction Glue Logic (TGL) block, the Remote Memory Segment Table (RMST) is consulted to identify if and where the request should be forwarded to. The RMST is a custom module (aimed to be implemented in custom logic on the MPSoC Programmable Logic) that acts as a proxy for the remote nodes (memory or accelerator). When transmitting data to a remote node, the translator is responsible for receiving the request from the PU, decoding it to determine where it should be sent, and forwarding it to the proper outgoing high-speed port, which leads to a circuit-switched path that would have been already set up via orchestration procedures (cf. Section 5.3). Along with the corresponding output link, the RMST will also provide with other management information necessary (remote memory size, offset, etc.). The RMST is a fully associative structure managed by the orchestration tools.
tools and the local Operating System; however, entries in this structure identify large and contiguous portions of memory space, so we expect that a few tens of entries will be able to address very large amounts of disaggregated memory space.

Figure 5 shows how the translator converts local virtual memory addresses to remote physical addresses and vice versa. At boot time of a dCOMPUBRICK and among other tasks, the orchestration tools (a) divide the node’s visible memory space to local segments, each one associated with a running VM, (b) allocate a set of corresponding remote physical memory segments, adjusted with a fixed offset, and (c) configure the local TGL block by writing to it for each local segment its start address, end address and remote offset. When the local dCOMPUBRICK issues a store to a memory address that falls within the remote region, then the local address is adjusted by the TGL translation unit with the corresponding offset, then encapsulated with the data plus required network info, and transmitted to the network. On the reverse path, incoming data consist of the requested payload, the segment id, and the remote memory address. Based on the segment id, the translator subtracts the required offset and forwards the locally translated address with the payload to the Processing System of the local dCOMPUBRICK.

Figure 5 - The Remote Memory Segment Table (RMST) performs address translation for accessing remote memory and store receiving data. The remote offset field is used to translate the local address to the disaggregated (remote) memory module space. The RMST provides additional information such as the output link to send each request.
### 2.2.2 Memory Brick Architecture (dMEMBRICK)

![Figure 6 - dMEMBRICK architecture](image)

Figure 6 shows the memory brick (dMEMBRICK) architecture, which is a key disaggregation feature of dReDBox. It will be used to provide a large and flexible pool of memory resources which can be partitioned and (re)distributed among all processing nodes (and corresponding VMs) in the system. dMEMBRICKs can support multiple links. These links can be used to provide more aggregate bandwidth, or can be partitioned by the orchestrator and assigned to different dCOMPUBRICKs, depending on the resource allocation policy used. This functionality can be used in two ways. First, the nodes can share the memory space of the dMEMBRICK, implementing essentially a shared memory block (albeit shared among a limited number of nodes). Second, the orchestrator can also partition the memory of the dMEMBRICK, creating private “partitions” for each client. This functionality allows for finer-grained memory allocation. It also requires some translation and protection support in the glue logic (transaction glue logic block) of the dMEMBRICK.

The memory translator logic interfaces with the requesting dCOMPUBRICKs, both of which are coordinated by the orchestrator software. Besides network encapsulation, the memory translator under the control of the orchestrator controls the possible sharing of the memory space among multiple dCOMPUBRICKs, enabling support for both sharing among and protection between dCOMPUBRICKs. The control registers allow the local mapping of external requests to local addresses to allow more flexible mapping and allocation of memory.

A dMEMBRICK can be dimensioned in terms of memory size as well as on the number of memory controllers it supports, so as to adapt to the size and bandwidth needs at the tray and system levels. In the scope of the project and due to constraints posed by the logic available on the cutting-edge FPGA module that is to be used for prototyping the dMEMBRICK (Xilinx Zynq Ultrascale+ MPSoC), we plan to implement up to 2 (two) memory controllers on the dMEMBRICK. Also, in reference to Figure 6, we note that the dMEMBRICK architecture is not limited to a specific memory technology, as long as this is supported by the transaction glue logic implementation. In the scope of the project, prototyping and evaluation is planned using DDR4 memory modules. Figure 6, we note that the dMEMBRICK architecture is not limited to a specific memory technology, as long as this is supported by the transaction glue logic implementation. In the scope of the project, prototyping and evaluation is planned using DDR4 memory modules.
### 2.2.3 Acceleration Brick Architecture (dACCELBRICK)

dACCELBRICK hosts the accelerator modules that can be used to enhance application execution based on the near-data processing scheme [3]; instead of transmitting data to a remote dCOMPUBRICK, certain calculations can be performed by local accelerators, thus improving performance and at the same time reduce network utilization. In fact, the project leader IBM, recently presented the ConTutto reconfigurable platform for near-memory acceleration, used in the memory subsystem of the OpenPOWER processing platform [1] [2].

The accelerator modules can range from custom hardware IPs to high-end GPUs (either soft-core or hard-core). In the scope of the project, and as depicted in Figure 7 and Figure 8 depict the dACCELBRICK architecture with and without local memory modules, we aim at showcasing acceleration capability made possible by the SoC to be used for materializing the dACCELBRICK; as with the dMEMBRICK, the local NI module is responsible for transmitting / receiving packet-based data to / from remote nodes, while circuit-based data are forwarded to the local interconnection module. For dBRICKs with local memory, the memory translator logic again provides memory translation and protection. Finally, the compute node will be able to (a) configure all available accelerators by writing to a predefined set of memory mapped registers, and (b) monitor their status by reading another set of memory mapped registers.

---

**Figure 7 - dACCELBRICK architecture integrating local memory**
2.3 Tray Architecture (dTRAY)

dTRAYs may be composed of arbitrary combinations of the three different types of dBRICKs detailed above. A dTRAY may have standard 2U size and may contain up to 16 bricks. It is expected that the number of dMEMBRICKs will be larger than the number of dCOMPUBRICKs and dACCELBRICKs, since a dCOMPUBRICK is expected to access multiple dMEMBRICKs. The different dBRICKs are interconnected among each other within the dTRAY and also with other dBRICKs from different dTRAYs. Figure 9 illustrates the dTRAY architecture.

Four different networks, one low-latency high speed electrical network, one Ethernet network, one low-latency high speed optical network, and one PCIe network will provide connectivity between the different bricks.

Accessing remote memory will use both optical and electrical low-latency high-speed networks. Accesses to remote memory placed in a dMEMBRICK within a dTRAY will be implemented via an electrical circuit crossbar switch (dBESM in Figure 9 is labelled as High Speed Electrical Switch) and will interconnect directly to the GTH interface ports available on the programmable logic of the bricks. The dBESM switch will have 160 ports. This is the largest dBESM switch available on the market today supporting our speed requirements. The latency will be as low as 0.5ns and the bandwidth per port will be limited to 12Gbps. This network will be used for intra-tray memory traffic between different bricks inside the tray. dBESM will not be used for inter-tray memory traffic due the limitations of the electrical communication in larger distances (latency). In addition, using electrical network for intra-tray communication instead of an optical network would not require signal conversion from electrical to optical and vice versa and thus it will be lower latency and lower power consumption.
Figure 9 – Sample tray architecture with multiple bricks interconnected through optical and electrical interconnection networks. All brick types are based on the Xilinx Zynq Ultrascale+ MPSoC.

The optical network is aimed to provide inter-tray connectivity for exchange of memory data and I/O communication. The optical network on the dTRAY will be implemented with optical switches (dBOSM in dReDBox terminology) that will be provided by Polatis. 3 optical switches will be places inside the dBOX. Each dBOSM switch will have 96 optical ports and total of 288 ports will be available for interconnecting the bricks in the same dBOX. The latency of the dBOSM optical switch would be around 5ns and the bandwidth would be at the range of 100Gbps. dBricks will connect to the dBOSM via GTH interface ports available on the programmable logic of the SoC. The GTH bandwidth is 16Gbps. A total of 24 GTH ports will be available in the SoC, 8 of them will be used to connect the SoC to the dBOSM. On a fully populated tray hosting 16 bricks, a maximum of 256 optical ports may be used to fully interconnect the bricks of each tray. The Mid-Board Optics (MBO) device mounted on each dBRICK will be used to convert the electrical signals coming from the GTH ports and aggregate them into a
single fiber ribbon; the other end of the ribbon will be attached to local dBOX’s dBOSM optical switch. Each MBO supports up to eight ports.

The dBRICKs will use the 8 GTH ports to connect to dBOSM. The number of GTH’s per SoC connecting to the dBOSM is limited by the size of the dBOSM. A 160-port dBOSM could support at maximum 10 GTH per dBRICK, given 16 dBRICKs on a tray.

The Ethernet (ETH) network will be used for regular network communication and board management communication (BMC). The bandwidth will be 1Gbps and it will have a hierarchical topology.

Inside the tray and across trays within a dRACK, the dBRICKs will also interconnect via PCIe interface. Such interconnection will be provided via a PCIe switch, which will exit the tray with one or more PCIe cable connectors. The PCIe interface will be used for signalling and interrupts, as well as for attachment to remote peripherals. This network can also be used to (re)configure the FPGAs in each SoC.

2.3.1 Power Consumption Estimation

The upper bound power consumption for a 2U dTRAY is expected to be in the order of 750W. The expected power consumption of a dBRICK is expected to be in the order of 20W. These power numbers are conservative and an early estimation. More detailed estimation will be described in next iterations of this deliverable, as well as in related deliverables from Work Package 5.

2.4 Rack Architecture (dRACK)

Figure 3 has already introduced the high-level dRACK architecture of the dReDBox project. Multiple dTRAYs of different configurations can be placed in the same dRACK. These dTRAYs can feature a different balance between Compute, Memory and Accelerators. dRACKs would be sub-divided into dCLUST. The decision of sub-dividing a dRACK into dCLUSTs is mainly motivated by a constraint that the largest dROSM will not be able to interconnect all the optical links from the dTRAYs. Besides this constraint, sub-rack based organization would also facilitate and simplify the locality management. A dROSM is placed at the top of all racks to provide inter-rack communications (dBRICKs in one dRACK to dBRICK in another dRACK).
3  dReDBox Hierarchical Network Architecture

3.1 System-level operation

In this section, we present an overview of the operating principle and network features of the proposed architecture. The architecture supports hybrid switching and multiplexing techniques, which include: optical circuit switching (dBOSM), electrical packet switching (EPS) and electrical circuit switching (dBESM). These hybrid techniques allow for reconfigurable topologies and dynamic functions for different traffic requests. Figure 10 presents an overview of the proposed architecture with some examples of reconfigurable topologies and functions using OCS/EPS/ECS.

This dRACK architecture, creates an end-to-end theoretical latency map under different transmission requirements using different assumptions. Figure 11 presents the analysis of theoretical end-to-end latency when using DDR4 memory modules on the dMEMBRICK, while Figure 12 presents the theoretical end to end latency with HMC memory modules on the dMEMBRICK.
Each dTRAY may contain dCOMPUBRICK, dMEMBRICK or dACCELBRICK bricks or any combination thereof. In each case, Remote Memory Segment Table (RMST) block would be implemented as glue logic in FPGA. In the case, where both types of memory modules are in use (DDR4 and HMC), a map table needs to be maintained which should be under control of hypervisor running on the PU of a dCOMPUBRICK to differentiate which type of memory (DDR4 or HMC) has been requested (cf. Section 2). This block is also responsible to generate custom network packets (includes physical addressing of the addressed dMEMBRICK) for remote memory and route out of the dBRICK using dedicated network port(s). This leads to lightweight proprietary “Memory over Network” protocol development featuring HMC and DDR4 mapped in same address space. We may introduce global unique brick IDs or hierarchical unique brick IDs as network addresses. These network IDs are important when we are accessing remote memory through packet switching. The extension of the “Memory over Network” protocol needs be to implement in dMEMBRICK, which will work as responder to the same block in dCOMPUBRICK. A separate HMC/DDR4 controller would be implemented on memory brick. It is important to mention here that NoC implemented in dMEMBRICK may differ with the same in dCOMPUBRICK however, it comes with the
same Layer1 electric switch connectivity.

### 3.2 Intra-tray connectivity (within dTRAY)

The approach for dTRAY architectural design is presented in Figure 13. It presents a heterogeneous tray architecture design composed of dCOMPUBRICK, dMEMBRICK with optical switches (dBOSM), electrical switches (dBESM), PCIe switch and Ethernet switch for hybrid optical and electrical network interconnect. This architecture provides both port-to-port optical circuit switching (OCS) and electrical circuit switching (ECS) network interconnect. This approach eliminates optical-electrical-optical (OEOP) conversion and reduces the latency for dBRICK to dBRICK intra-dTRAY communication. Note that each dBRICK only uses one FPGA or optical/electrical mid board to connect to both the optical and electrical switch. Due to the size and port dimension of currently available dBOSM, the dBOSM switch(es) are planned be placed outside the tray and within the dBOX (we remind that dTRAY is like motherboard placed in dBOX).

![Figure 13 - Heterogeneous dTRAY architectures with hybrid optical and electrical network interconnecting bricks within a dTRAY (applying to dACCELBRICKs too, albeit not shown)](image)

Different Configurations of dTRAY architectures can be realized by varying; number of dBRICKS in a dTRAY, port dimension configuration per dBRICK, number of dBOSMs in a tray and port dimensions of dBOSMs in a dTRAY.

### 3.3 Inter-tray connectivity (between dBOXes)

Figure 3 presents our proposed dRACK scale architecture with heterogeneous dTRAY architecture that includes hybrid optical and electrical interconnect. The dRACK scale architecture comprises of different stacks of heterogeneous dTRAYs connected to optical switches (middle of rack) placed at the middle of the stack of dTRAYs in the
dRACK. The optical switches are in the middle of dRACK because this approach reduces length of fibre required to connect the dTRAYS to optical switch for inter-dTRAY communications. This leads to a reduction in latency and fibre length when compared to the traditional server-centric datacenter architectures with Top of Rack (ToR) switches. In addition, functional modules such as packet switches can be easily plugged to the dROSM to provide dynamic functionality for processing traffic requests for inter tray and inter rack communication. A top of cluster (dDOSM) optical switch is placed at the top of all dRACKS to provide inter-dRACK communications (dBRICK in one dRACK to dBRICK in another dRACK).

In addition, by varying number of dBOSMs, port configuration of dBOSMs, numbers of dROSMs and port connectivity ratio (inter dTRAY traffic to intra dTRAY traffic) used in the dRACK scale architectures design, different configurations of dRACK scale architectures are achieved.

### 3.4 Cabling Complexity

The type of cables used will depend on the dTRAY and dRACK level network interconnect design; mode of physical link (electrical or optical), number of channels a link support, number of MBO per dBRICK, number of port on electrical or optical cross bar switch and type of multiplexing techniques fiber supported. The optical interconnect can use a fiber ribbon with 8 fibers equivalent to the 8 number of channels of the MBO. For inter-dTRAY and inter dRACK communication which is pure optical network interconnect, the number of fibers ribbons will depend on the port dimensions of the optical switches (dTRAY, dRACK and dCLUST level), hierarchical port connectivity from the EOT optical switches to MOR optical switches and inter-dTRAY and from MOR optical switches to TOC optical switch for inter-dRACK.

A comparison between optical and electrical connectivity is shown in Table 1.

<table>
<thead>
<tr>
<th>Optical switch/MBO</th>
<th>Electrical switch/Connectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fiber has less power consumption</td>
<td>OEO conversion introduce additional power consumption</td>
</tr>
<tr>
<td></td>
<td>OEO conversion needs additional space</td>
</tr>
<tr>
<td>Manageable capex when maximum connectivity needed</td>
<td>Optical switch and transceiver/MBO increase the Capex</td>
</tr>
<tr>
<td>Increased I/O panel density</td>
<td></td>
</tr>
<tr>
<td>Optical signal can be MUX/DEMUX by SDM or WDM technology</td>
<td></td>
</tr>
<tr>
<td>Bandwidth per port is not limited</td>
<td></td>
</tr>
<tr>
<td>Longer reach (suitable for Inter-tray Communication or inter-rack communication)</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 - Comparison between optical and electrical connectivity
4 Scalability Analysis and Considerations

This section analyses the main factors that govern the scalability in the number of components of the system. Specifically, it will analyze the scalability of the number of dBRICKs, dTRAYs, and dCLUSTs, and dRACKs.

4.1 dBRICK scalability

The scalability of the dBRICK in terms of number of computing processors is dominated by the physical dimensions. Figure 14 shows the physical dimensions of the dBRICK. As you it is shown, dBRICK has a length of 180mm and a height of 67 mm. The Xilinx Zynq UltraScale+ EG MPSoC ZU19EG has a dimension of around 35mmx35mm, one SODIMM memory module has a dimension of around 60mmx30mm. Therefore, only one Zynq UltraScale+ EG MPSoC could be placed in a dBRICK.

![Figure 14: dBRICK dimensions](image)

The number of GTH ports available in the Zynq UltraScale+ EG MPSoC are 16 or 24 depending on the model. The Zynq UltraScale+ EG MPSoC XCZU9EG-2FFVB1156E has 24 GTH ports, whereas the Zynq UltraScale+ EG MPSoC XCZU9EG-2FFVC900E has 16 GTH ports.

Regarding the number of MBO devices (Luxtera LUX62608) that can be placed in a dBRICK, these could be one or two. In the case of placing two MBOs, then the dBRICK is bigger forcing to use a 3U dTRAYs instead of the 2U housing standard as preferred.

Regarding the number of memory SO-DIMMs, two or four SO-DIMM memories could be placed in a dBRICK. However, four SODIMMs need a bigger dBRICK that forces to use the 3U dTRAYs. On the other hand, two SO-DIMMs could be used in a 2U dTRAY.

4.2 dTRAY scalability

This section analyzes the scalability of the dTRAY in terms of the number of dBRICKs that could be allocated in a dTRAY.

The dimensions of a standard 2U dTRAY are width 40cm, length 45 cm, and height 75mm. The maximum number of dBRICKs that can accommodate a dTRAY is sixteen based on the physical dimensions of the dBRICKs and dTRAY. In the unfortunate (but improbable) case during prototyping of running out of floorplan budget on a dTRAY due to space needed by other components needed (such as the power supply or forced use of switches of other dimensions), the number of dBRICK slots per tray could be reduced down to twelve.
4.3 dBOX scalability

The dBOX will support various number of internal optical switches (dBOSM) to provide connectivity to other dBOXes in rack. Due to space limitations in a dBOX (see Figure 15), the maximum number of dBOX that could be placed would be three.

The selected dBOSM will be the Polatis Series 6000 single-sided 48xCC OSM that provides 24 bidirectional ports. The dimensions of this device are 41mm x 122mm x 266mm. This is the most compact optical switch module available from Polatis and thus this was the reason to select this switch. In the near term, Polatis is also developing the next generation of OSM which will do 96 ports in the same form factor as the 48xCC OSM.

With three dBOSMs, the total number of bidirectional ports in a dBOX would be 144. 144 bidirectional ports would allow 1:1 subscription (this is full intra and inter tray connectivity) for up to 9 bricks. Also, in case of 16 bricks per tray, 144 bidirectional ports would allow connecting all 16 bricks to bricks on different trays with 1:2 oversubscription (this is 4 bidirectional ports per tray or half rate).

Figure 15: dBOX maximum dimensions in millimeters
4.4 dRACK scalability

Figure 16 shows the standard full size rack of 42U, with 1.8m height. A dRACK will consist of the dROSM and several dBOXes. The height of a dROSM would be 4U whereas each dBOX would be 2U. Therefore, the total number of dCLUST in a dRACK would be 19.

Assuming that each dBOX will allocate at maximum 16 dBRICKs then the total number of dBRICKs in a dRACK would be 304 (19x16).

Figure 16: Dimension and components in a dRACK

4.5 Datacenter scalability

This section analyses the maximum number of dRACKS in a datacenter. Figure 17 shows a possible solution for interconnecting the different dRACKS in a datacenter. It consists on using a higher-level OSM (dDOSM) on top of the existing dROSM in order to interconnect every different dRACK in a datacenter. Currently, the biggest dDOSM from Polatis is the series 6000 with 384 bidirectional ports OSM. Every dRACK can connect to the dDOSM using one or more fibers (n). Table 2 shows the resulting maximum number of dRACKs and corresponding dBRICKs that can be supported by the largest dDOSM today. It can be seen that the largest datacenter could allocate up to 116,736 dBRICKs (16 dBRICs x 19 dTRAYs x 384 Dracks = 116,736 dBRICKs). Notice that in this case, it is assumed an over-subscription network where only one
D2.3 – System Architecture specification (a)

dBRICK from each dRACK could communicate to another dBRICK in another dRACK. This is an extreme case that provides maximum system scalability. On the other case, dReDBox does not limit the amount of fibers coming out from the dRACKs. There could be other possible system configurations where the number of fibers per dRACK could be set up larger. For illustration purposes, Table 1 shows the case when there are 8 fibers coming out from the dRACK, and in this case, the maximum number of dBRICKs will be reduced to 14,592 (16 dBRICKs x 19 dTRAYs x 48 dRACKs = 14,592 dBRICKs). It is calculating by dividing up by 8 the previous numbers.

![dRACKs interconnection in a data center](image)

**Figure 17: dRACKs interconnection in a data center**

<table>
<thead>
<tr>
<th>Number of fibers per dRACK (n)</th>
<th>dRACKs</th>
<th>dBRICKs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>384</td>
<td>116,736 (16 dBRICKs_per_tray x 19 dTRAYs_per_rack x 384 dRACKs)</td>
</tr>
<tr>
<td>8</td>
<td>48</td>
<td>14,592 (16 dBRICKs_per_tray x 19 dTRAYs_per_rack x 48 dRACKs)</td>
</tr>
</tbody>
</table>

**Table 2: Data center sizes**

5 Orchestration tools and System Software Architecture

This section describes the System Software components that will be developed by the project in order to manage and orchestrate the dReDBox hardware, and exploit the benefits of its disaggregated architecture.
This chapter is organized as following. Section 4.1 explains how the concept of Software Defined Memory (SDM) is reflected in dReDBox system software, and Section 4.2 extends this description by exemplifying the main ideas behind the software-defined memory allocation and deallocation. Finally, Section 4.3 provides a more detailed architectural-level description of the main System Software components, while Section 4.4 concludes the discussion by describing a software-based mechanism to implement shared memory on top of the dReDBox architecture.

5.1 Rack-scale software defined memory system architecture

dReDBox is a vertical disaggregated architecture that requires hardware and software synergy to implement resource pooling. In the spirit of the Software Defined Infrastructure (SDI) vision, dReDBox aims at an architecture that will allow authorized tenants to describe in software and reserve system resources, at a fine-grained level, to execute their applications. Accordingly, the dReDBox system adopts the virtual machine as the execution container and type-I Virtual Machine Monitor (VMM) architecture to support a software defined deployment. Therefore, the challenges that need to be addressed in the context of the dReDBox orchestration and system software architecture are the following:

a) Design and development of a fine-grained resource reservation support and appropriate interconnect configuration subsystem.

b) Global Memory address space management and allocation/freeing mechanisms at all involved layers.

c) Virtual Machine Monitor (VMM) memory subsystem enhancements to facilitate memory hot plugging as well as to integrate remote memory in the NUMA hierarchy and possible page relocation policies to improve execution performance.

d) VMM-based memory access control for prohibiting accesses to physical memory regions that are not currently/yet interfaced to a physical memory resource.

e) Architectural changes and enhancements that are related to the ability to dynamically change, at runtime, the physical hardware platform configuration. VM migrations will be also addressed in this context.

5.2 Memory allocation and deallocation processes

From the point of view of a dReDBox User, Virtual Machines (VMs) are the principal unit of resource allocation; users request VMs with a selected number of virtual CPUs (vCPUs) and amount of main memory, and the System Software will transparently set-up the underlying hardware infrastructure and allocate the resources needed to satisfy user requirements from memory bricks to compute bricks, and finally start the user VMs.

We envision two distinct but similar processes through which memory is allocated to VMs and, consequently, to compute bricks:

a) at VM initialization and deployment, and

b) at VM runtime to respond to dynamically changing memory requirements.

The dReDBox orchestration component receives VM allocation and deallocation requests from users. This request will contain at least three required parameters:

1. Number of vCPUs to allocate to the VM
2. Amount of guaranteed memory to allocate to the VM
3. Maximum amount of memory for the VM

While the meaning of the number of vCPUs is quite obvious, the differentiation between guaranteed and maximum memory is less straightforward. Guaranteed
memory is the minimum amount of memory that the VM will be assigned at any point in time, independently on its actual memory consumption. At the same time, a VM is allowed to expand and shrink its currently allocated memory beyond the guaranteed limit, based on dynamic usage; the maximum memory value controls the amount of memory that a VM is allowed to request, subject to global availability.

At VM initialization and deployment time the orchestration subsystem determines the compute brick on which the VM should run, reserves the requested guaranteed memory, and sets the rack interconnect up so that the selected compute brick can access that memory, before letting the system software running on the compute brick deploy and start the VM.

At runtime, the VM can expand its memory up to the maximum memory threshold. To do so, dReDBox will build on already existing memory ballooning techniques and adapt them to the disaggregated architecture. In more detail, the ballooning driver will interact with the VM hypervisor and the local dReDBox software and, in coordination with the global orchestration tools, it will allocate and set-up additional memory from the remote pool. A similar process will be executed in case of reduced VM memory pressure to dynamically return unneeded memory.

Finally, when a VM is stopped the memory is also returned to the global memory pool, although the association with the compute brick might or might not be canceled immediately, depending on optimization policies. The process described in the this section is at the core of the dReDBox elastic resource management functionalities and will be implemented by a set of distributed software components.

### 5.3 Orchestration tools high-level architecture

![Orchestration tools high-level architecture](image)

Figure 18 - Orchestration tools high-level architecture
In this Section, we will provide an overview of these components, emphasizing on the main design decisions for what concerns their role and the interplay among each other. Figure 18 shows a coarse grain architectural overview of the dReDBox system software stack, and more detailed descriptions of individual software modules are provided in the following subsections.

5.3.1 Orchestration tools

The orchestration tools are the set of software modules that coordinate to implement two main functions:

1. Manage and keep the state of the software defined memory abstraction;
2. Expose an IaaS-like interface to dReDBox users to let them create and deploy VMs on the hardware infrastructure.

In our architecture, the orchestration tools are defined by the following components (Figure 19)

- **IaaS Scheduler.** This component will be the entry point for the user to inspect the status of the dReDBox infrastructure, make VM allocation requests and also manage her VMs. Our goal is to offer an intuitive web-based dashboard to the user, through which she could spawn new VMs and monitor their status. Under the hood, this component aims to define REST endpoints that would power the Web UI and will also interact with the Software Defined Memory Controller for requesting reservations on the compute and memory bricks. IaaS Scheduler keeps record of the assignment of compute bricks to the VMs so that when a new VM allocation request arrives it is capable of defining a list of preferred compute bricks based on the policies and the constraints defined.

- **Software Defined Memory Controller (SDM-C).** SDM-C is the component responsible for allocating and de-allocating memory to compute bricks. At VM initialization, it receives allocation/deallocation requests from the IaaS
Scheduler. SDM-C aims to maintain internally the status of the memory bricks when it comes to free and occupied address ranges and, based on this, to satisfy allocation requests. SDM-C will also be responsible of taking into account the hardware constraints of the underlying platform (e.g., maximum number of circuit-switched optical connection from a compute brick to memory bricks). Moreover, SDM-C will interact with the Software Defined Memory Agent, which runs on the compute bricks. At VM initialization and deployment time, SDM-C will notify the SDM-A of the memory allocation decisions (which memory bricks to use, alongside with the corresponding address ranges), while, dynamically, SDM-A and SDM-C will interact for memory resizing based on dynamic pressure measurements.

- **Software Defined Memory Agent (SDM-A).** SDM-A would be running on the compute bricks and is responsible to implement local management of remote memory allocation and deallocation. Its actions are instructed by the SDM-C. Moreover, SDM-A will send periodic messages (“heartbeats”) to SDM-C to communicate its health state.

### 5.3.2 Compute brick system software

Each compute brick exports a memory layout, hardware virtualization and interrupt facilities to bootstrap a proper operating system, see Figure 20. In the context of dReDBox, Linux kernel will be used to take over of the bare metal resources. The kernel will require some modifications to properly take advantage of dReDBox compute brick and the ability to access remote memory.

Among the most important components will be the dReDBox Memory Management layer which will handle the dynamic mapping of remote memory regions. This subsystem will also facilitate performance enhancements (e.g., transparent page migration to local memory for frequently accessed pages) as well as coherent transitions from one memory configuration to another.

The SDM kernel driver will be used by orchestration tool agent to configure on the remote memory controller the segment address translation and interconnect network information.

The KVM hypervisor will be used to execute the user workload in the form of virtual machines and will be appropriately extended where needed to cooperate with the dReDBox management layer. VirtIO support will be appropriately extended to control both KVM and memory management layer.

On the main system user space, an appropriate orchestration tool agent (SDM-A) will be developed that will coordinate the remote memory distribution and support the software defined platform synthesis concept.
5.3.3 Virtual machine management

Like most current solutions, a crucial component is the virtualization layer running between deployed workloads and available hardware. It benefits the system in manifold ways, for example by providing more flexible resource allocation and therefore greater system utilization density. Additionally, virtualization provides increased security level by VM isolation and offers additional fault tolerance mechanisms like checkpointing and migration. Nevertheless, the dReDBox project innovative design assumes physical disaggregation of multiple compute nodes from multiple memory nodes in order to achieve its goal of decoupling the amount of available memory from CPU resources. Because of that, the main challenge regarding virtualization software is its modification that will provide virtual machines, deployed at multiple compute nodes, with remote resources in an efficient way.

The virtualization solution used in dReDBox is an open source Linux-based software called KVM [4]. KVM (from Kernel-based virtual machine) is actually a Linux module that provides virtualization infrastructure, therefore turning an operating system into hypervisor. By itself, KVM is not responsible for running any virtual machine, it just exposes a user-space interface, in form of a device, and is interacted by a user-space virtualizer, namely QEMU [5]. Thanks to that, it remains relatively compact in terms of code size, performing only those operations that cannot be moved to user-space and limiting the number of changes affecting system kernel. QEMU, on the other hand, is a user-space application, here serving as a virtualizer (however QEMU by itself can also run in other modes), that is running virtual machines and providing hardware virtualization features.

Because of the disaggregated dReDBox system architecture, when the majority of memory as well as devices are placed remotely (with regards to computation node running a hypervisor), the aforementioned virtualization software (QEMU) has to be
modified in order to become capable of exploiting all resources globally shared over the system. In its current state, when QEMU launches a virtual machine a given amount of locally available memory is allocated and passed to the guest operating system. However, in dReDBox design, the amount of locally available memory is minimal and destined for hypervisor purposes, thus all memory used by VMs should come from remote nodes. This novel approach requires appropriate modifications in QEMU, so that a memory segment is fetched from global pool by the host before being offered to a guest system.

Global memory resources reservation should be done in a coherent way so that no two guests will use overlapping regions, which would lead to memory corruption (memory sharing is an exceptional case). Therefore, the hypervisor should interact with system orchestrator, which keeps a global memory mapping. At host-level it has been proposed to implement global memory reservation stage either by reusing ballooning technique or by memory hot-plug.

Memory ballooning (Figure 21) is a technique currently used between host and guest, where the balloon is a device that owns a given amount of system memory pages and is capable of passing them to guests (deflate) in a flexible way, once requested. Respectively, the balloon can also reclaim the pages back (inflate) from running guests. Nevertheless, in the dReDBox project this pattern is going to be adapted for the purpose of obtaining and releasing memory pages from global memory pool performed by the hypervisor. Thanks to that the amount of memory allocated for a given computation node can be dynamically adjusted according to the workload demands and system policy.

A similar technique, from high-level perspective, is memory hot-plug that also allows changing the amount of available memory in a dynamic way. In essence, it makes the running kernel aware about conducted hardware reconfiguration and trigger its internal structures (e.g. virtual memory map and page table) update so that a virtual address space can be enlarged or shrank without bringing the computation node down. In dReDBox, from the hypervisor perspective, hardware reconfiguration could mean fetching or releasing memory resources from the global pool. Memory hot-plug is currently available in the Linux kernel for a limited set of architectures (including x86, Itanium and SPARC) to allow increasing/decreasing the available memory from locally attached memory modules. In the context of the project, we will need to extend support to remote memory and possibly to the processor architecture selected by dReDBox (ARMv8).

After remote memory reservation, the hypervisor is ready for launching a VM. This is done with overcommitted memory, which means that the hypervisor claims to assign

Figure 21 - Using memory ballooning, a balloon device owns a given amount of memory and is capable of mapping pages from and to virtual machines according to current needs.
a VM with a large amount of memory, whereas only a part of it is actually backed by resources obtained from the global pool. Remaining part consists of kind of void addresses and they need to be assigned with an actual memory only when first access will be detected by hypervisor.

This design does not prevent using the classical ballooning technique (between host and guest - see Figure 21) on top, for flexible balancing available memory between multiple virtual machines within one computation node. However, this is an already existing solution and will not be discussed deeper here.

### 5.4 Shared memory and device virtualization

Additional challenges coming with dReDBox architecture are global memory sharing and remote devices access sharing. Both problems are very similar and so the proposed solutions.

In case of classical virtualized environments, one way of sharing locally attached devices between QEMU virtual machines is by using VirtIO, see Figure 22. VirtIO is a device drivers framework based on para-virtualization, that is splitting the driver into two parts; front-end installed in the guest and back-end installed in the host. The former exposes an interface expected by guest operating systems but does not directly orchestrate the corresponding device. Instead, it communicates with the latter, which is installed in the hypervisor and talks to the hardware. That way, one device can be simultaneously shared amongst multiple guests. A crucial component of the framework is the communication channel, that is composed by unidirectional queues (called virtqueues) located in a memory area shared between host and guest. Buffers of data are exchanged by taking turns. First, a queue is filled with them and then, the other end is signaled, so that these buffers are received. After fetching the data, the sending side is also notified in order to re-enable the queue. It has been proposed to address both problems of accessing shared memory and remote devices with an approach based on VirtIO operational design but with both communicating ends located in different system nodes.

![Figure 22 - Example VirtIO driver architecture for a PCI device. Vring is located in a shared memory accessed by both front-end (guest side) and back-end (host side).](image)

Previous paragraphs have described remote memory access, including a reservation stage. For sharing a certain memory region between multiple virtual machines all of this remains valid, however there are additional requirements present. Firstly, the guest should be able not only to ask for any chunk of memory but it should explicitly specify that this global memory segment is going to be shared. As a response, it should obtain information serving as handler, so that any VM using this handler is also able to access shared memory region. Secondly, there is a control channel required to pass the handler between sharing peers and implements access synchronization as well. Synchronization here means notifying (by using a provided API) the other peer about
availability of new data and passing access right to it in the same way. The exact way
how this channel will be created will be defined later but two preliminary proposals are
either over a control network (using software-only means) or PCI-Express-based rack-
scale message passing mechanism (called mailbox, described below). The former will
naturally offer lower performance but is perhaps simpler to implement whereas the
latter depends on provided hardware features and this is not known yet at the current
stage of the project.

Figure 23 - Remote device access high-level scheme based on modified virtio technique.
Data passed through virtqueues located in shared memory and the other end notified by
available signaling means.

Similarly, in case of disaggregated peripherals, the dReDBox architecture assumes
that they are not attached locally to the computation nodes but they reside in remote
ones. Therefore, the access method is based on shared memory with yet additional
requirements.

A driver back-end will be installed on the node that has a peripheral installed locally
and front-end will remain in the guest operating system. To make the framework
operations possible, the virtqueues for sending buffers should be allocated in the
global memory shared by both sides and a separate channel for passing control
messages between front-end and back-end is necessary (see Figure 23). These two
things boil down to the same requirements as in shared memory case. But an
additional issue that needs to be handled is the knowledge about available peripherals
and corresponding virtqueues location. In a first approximation, the concept assumes
that the front-end allocates corresponding virtqueues during bootstrap in global shared
memory and initializes corresponding back-end. Naturally, that requires mediation of
the orchestrator for a shared access setup and peripheral initialization. The exact
interaction scenario has not been design yet.
6 Implementation considerations

This section briefly presents a set of first implementation considerations regarding the dReDBox architecture described above. D5.1 will provide finer level of detail and final implementation decisions.

- **dReDBox networks:** As mentioned, a dReDBox system will require a reliable network for the orchestration tools, a data network for transfers among compute / memory / accelerator bricks, and a network for delivering remote interrupts. A first consideration is to use a packet-based Ethernet network for the orchestration tools, a circuit-based network for low-latency data transfers, and finally a separated PCIe network for serving remote interrupts. The packet-based network may utilize a Gigabit Ethernet port per brick, all connected to a level 2 Ethernet switch at the tray. In addition, the data plane network will utilize electrical switches for intra-tray communications, and optical switches for inter-tray communication. However, a decision that will be further explored is whether to “reserve” a certain number of the optical links for the packet-based network.

- **Memory brick:** A first suggestion is that each memory brick hosts at least 64 GBs of DDR4 RAM. Constrained on resources, however not in the critical path of the project, other memory technologies may also be considered (e.g. HMC), since they are already supported by major vendor tool providers.

- **Brick prototypes:** while exhibiting different functionality and carrying distinct roles in the dReDBox datacenter, the prototypes of the three dBRICK types are aimed to derive from the same brick SODIMM card design integrating the Xilinx Zynq Ultrascale+ MPSoC. This reduces the risk of design/prototyping flaws and also keeps the prototyping within resources. Obviously, the prototypes will also deviate in terms of on-board components, depending on the brick type (e.g. number of DDR4 memory modules).

- **I/O tray:** In order to support the connection of other external devices, the dReDBox will use a COTS PCIe I/O tray. The latter will allow other devices, such as SSDs / HDDs, network cards, to be connected.
7 Conclusions

This deliverable introduced the dReDBox disaggregated system architecture. The basic building unit in dReDBox is called brick. A brick will be a standard pluggable module that provides part of the system functionality. dReDBox will consist of three different types of bricks: compute bricks, memory bricks and accelerator bricks.

Multiple bricks will be interconnected through multiple interconnects used for different purposes. This is a significant departure of today’s data center architectures where the main building block is the mainboard with a fixed amount of resources (compute, memory, and accelerator) and accessing main memory through a parallel interface.

The dReDBox architecture overcomes this server limitation, significantly improving the resource utilization and opening large-scale efficiencies. To realize this disaggregated architecture, a memory translator has been introduced in each brick to provide required support seamlessly for accessing remote bricks. This is a key component of the architecture that glues together the different bricks (compute, memory, and accelerators) in the data center. This memory translator will be a custom reconfigurable logic that it could be tailored to interface any kind of processor, memory, and accelerator device.

Finally, another key component of the architecture is system software that manages and orchestrates the dReDBox hardware, and exploits the benefits of its disaggregated architecture. System software will transparently set-up the underlying hardware infrastructure and allocates the resources needed to satisfy user requirements from memory bricks to compute bricks. One of the key challenges resides on the virtualization software (QEMU) making it capable of exploiting all resources globally shared over the system. Additionally, it is considered an extension that will facilitate data sharing of remote memory between virtual. Memory sharing will be supported by the orchestration layer whereas device sharing will be implemented through Virtio and the support of remote interrupts through PCI.


8 References


This section analyses the pros and cons of two different commercially available processors candidates to be used in the dCOMPUBRICK. The selected processors are the Xilinx Zynq UltraScale+ EG MPSoC and the NXP LS2088 (NXP). Both processors are based on the low power ARMv8-A architecture.

The ARMv8-A architecture is well supported in the open-source community, and has many virtualization features which makes them attractive to be used in the dCOMPUBRICK. The main difference between them is that the Zynq UltraScale+ EG MPSoC provides a powerful programmable interface (FPGA) that could be used to access remote memory whereas there is no such feature in the NXP. A more detailed comparison between these processors is given in Table 3.

<table>
<thead>
<tr>
<th></th>
<th>Xilinx Zynq UltraScale+ EG MPSoC</th>
<th>NXP LS2088</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1 cluster of 4x ARM A-53 cores</td>
<td>4 clusters of 2x A-72 cores</td>
</tr>
<tr>
<td>Caches</td>
<td>32KiB L1 instruction cache per CPU</td>
<td>48KiB L1 instruction cache per CPU</td>
</tr>
<tr>
<td></td>
<td>32KiB L1 data cache per CPU</td>
<td>32KiB L1 data cache per CPU</td>
</tr>
<tr>
<td></td>
<td>1MiB L2 cache per CPU</td>
<td>1MiB L2 cache per CPU</td>
</tr>
<tr>
<td></td>
<td>1MiB L3 cache shared between all CPU cores</td>
<td>1MiB L3 cache shared between all CPU cores</td>
</tr>
<tr>
<td>CPU Frequency</td>
<td>Up to 1.5GHz³</td>
<td>2GHz</td>
</tr>
<tr>
<td>CPU pipeline</td>
<td>in order, 2 way, 2 dispatches max</td>
<td>out of order, 3 way, 5 dispatches max. (a72 128 rename registers/write back queue)</td>
</tr>
<tr>
<td>Cache pre-fetching</td>
<td>L1 data pre-fetcher</td>
<td>L1 data and L2 cache pre-fetcher</td>
</tr>
<tr>
<td>Maximum PA space</td>
<td>40-bit PA space =&gt; 1 TiB</td>
<td>44-bit PA space =&gt; 16 TiB</td>
</tr>
<tr>
<td>(Physical Address Space)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum VA space</td>
<td>48-bit (256 TB)</td>
<td></td>
</tr>
<tr>
<td>(Virtual address space)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cluster Interconnect</td>
<td>CCI-400</td>
<td>CCN-504</td>
</tr>
<tr>
<td></td>
<td>The CCN-504 has support for L3 Cache (1M for LS2088)⁴</td>
<td></td>
</tr>
<tr>
<td>SOC Cache Coherency</td>
<td>AXI ACE</td>
<td>CHI</td>
</tr>
<tr>
<td>interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMMU support</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Process size</td>
<td>16 nm</td>
<td>28 nm</td>
</tr>
<tr>
<td>Availability</td>
<td>Q2 2017 (GTH)</td>
<td>Q3/4 2016 (engineering samples)</td>
</tr>
<tr>
<td></td>
<td>Q4 2017 (GTY)</td>
<td></td>
</tr>
</tbody>
</table>

³ This is for the fastest speed grade. Numbers for the other speed grades are not yet publically available, but looking back at the previous generations, the mid-level and slowest speed grade will not support this CPU frequency.

⁴ The CCN-504 has better performance for multi-cluster systems, but since the ZYNC only has a single cluster, there is no advantage in this case.
The Zynq UltraScale+ EG MPSoC and the NXP have a comparable architecture. Both use the ARMv8-A instruction set, and they are binary compatible.

There are a lot of configuration parameters on the ARM system which a SOC implementer can change/tune. These options (where relevant), are described in this section, with a small explanation on what the differences mean with regards on performance of the system.

The Zynq UltraScale+ EG MPSoC uses the Cortex-A53 Processor core, while the NXP uses the Cortex-A72 Processor core. Both these cores are from the same generation of the ARM big.LITTLE™ architecture. The A53 core is the little core, and the A72 the big core.

The Zynq UltraScale+ EG MPSoC has one cluster of four A53 cores, whereas NXP has four clusters of two A72 cores. The four clusters allow for different cores running at different clock speeds, making fine grained CPU scaling possible. The ARM architecture does not allow for cores in the same cluster to run at a different frequency though.

The Zynq UltraScale+ EG MPSoC can run up to 1.5GHz, whereas the NXP up to 2 GHz. The 1.5 Ghz of the Zynq UltraScale+ EG MPSoC is for the maximum speed grade of the SoC, and it not clear at this time at which speed the other speed grades can operate.

The A53 and A57 have different pipe-line architecture. The A53 has an in order, 2-way, 2 dispatches maximum pipeline. The A72 has an out of order, 3-way, 5 dispatches maximum pipeline with 128 rename registers/write back queue. Also, the A53 has L1 data cache line prefetching, where the A72 has L1 data and L2 cache prefetching.

The maximum addressable Physical Address (PA) size of the A53 is 40 bit, where the A72 has a PA of 44 bit. This gives the A53 a PA space of 1 TiB, and the A72 a PA of 16 TiB. Both have a 48-bit Virtual address space (256 TB).

The Zynq UltraScale+ EG MPSoC has a CCI-400 Interconnect where the LS2088 has a CCN-504 Interconnect. The CCN-504 has a more performant cache coherency mechanism between clusters, but because the Zynq UltraScale+ EG MPSoC has only a single cluster there is not a real difference in this configuration. The only difference is that CCN-504 support a L3 cache, of which the NXP implements 1MiB.

The Zynq UltraScale+ EG MPSoC uses AXI ACE for cache coherency on the SOC, where the NXP uses CHI. CHI is the successor of AMBA 4 ACE technology, and has been developed to support high frequency, non-blocking data transfers between multiple fully coherent processors. The interface ensures optimal system performance by supporting distributed level 3 caches, very high rates of I/O coherent communication, and Quality of Service (QoS) functionality. Both devices have SMMU support (System Memory Management Unit), which helps out with virtualization.

<table>
<thead>
<tr>
<th></th>
<th>DMIPS</th>
<th>Core Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zynq UltraScale+ EG MPSoC @1.5 GHz</td>
<td>18,360</td>
<td>22,860</td>
</tr>
<tr>
<td>Zynq UltraScale+ EG MPSoC @1.2GHz</td>
<td>14,688</td>
<td>18,288</td>
</tr>
<tr>
<td>NXP @2GHz</td>
<td>97,600</td>
<td>81,600</td>
</tr>
</tbody>
</table>

Table 4: Performance comparison

From a performance and power perspective the ls2088 is the best choice, but the problem with this SOC is that there is not a readily available interface to connect to the...
D2.3 – System Architecture specification (a)

disaggregated resources. The nature of the dReDBox project, and the emphasis on the remote resources, together with a constrained budget has pushed the choice to the Zynq UltraScale+ EG MPSoC. The Zynq UltraScale+ EG MPSoC provides a programmable interface that can be used to interface to the remote resources, provide accelerator functions and allows for a tightly integrated dBRICK. This will help to efficiently develop the Transaction Glue Logic (TGL) and the Remote Memory Segment Table (RMST) in hardware.
10 Appendix II - Arrayed Waveguide Grating Router (AWGR) based solutions

Alternative networking scenarios employing an Arrayed Waveguide Grating Router (AWGR) passive module will be also explored within the frame of dReDBox project, as potential more forward looking alternatives to commodity optical switches. Among the existing optical interconnect architectures for datacenters, the arrayed waveguide grating router (AWGR) based solutions have attracted much attention due to WDM parallelism, dense interconnectivity and unique wavelength routing capability. An NxN AWGR is a passive wavelength router with N-input and N-output (N×N) in which input Wavelength Division Multiplexed (WDM) signals are de-multiplexed and cyclically processed to each output port according to their wavelength. Figure 24 below illustrates the basic operation of an 8×8 cyclic AWGR. In each input port a multiplexed signal can be injected that can be de-multiplexed to all the output ports. Each pair of input-output ports communicate through a different path that corresponds to a different wavelength. This cyclic wavelength arrangement is useful in order to avoid the wavelength contention.

In addition, the latency induced by an AWGR can reach up to a few hundreds of picoseconds (e.g. roundtrip time is approximately 600-700ps) depending on the dimensions of device rendering it as a suitable candidate for latency-sensitive applications. Currently, the largest scale of commercially available NxN AWGR routers has 32 input and 32 output channels and operates in C band. Based on such a device, multiple AWGR-based configurations will be investigated to address both the challenges and the benefits when an AWGR is employed for Brick-to-Brick or Tray-to-Tray communication.