TECHNISCHE UNIVERSITÄT DRESDEN

HPC - HIGH PERFORMANCE COMPUTING (SUPERCOMPUTING)

DISTRIBUTED OPERATING SYSTEMS, SCALABILITY, SS 2014

Hermann Härtig

Understand

- Systems Software for “High Performance Computing” (HPC), today & expected
- MPI as a common programming model
- What is “noise”?
- How to use incomplete information for informed decisions
- Advanced Load Balancing techniques (heuristics)

CLUSTERS & MPP

Characteristics of MPP Systems:

- Highly optimised interconnect networks
- Distributed memory
- Size today: few 100000 CPUs (cores) + XXL GPU

Successful Applications:

- CPU intensive computation, massively parallel Applications, small execution/communication ratios, weak and strong scaling

- Cloud ?

Not used for:

- Transaction-management systems
- Unix-Workstation + Servers

CLUSTERS & MPP

Characteristics of Cluster Systems:

- Use COTS (common off the shelf) PCs/Servers and COTS networks
- Size: No principle limits

Successful Applications:

- CPU intensive computation, massively parallel Applications, larger execution/communication ratios, weak scaling
- Data Centers, google apps
- Cloud, Virtual Machines

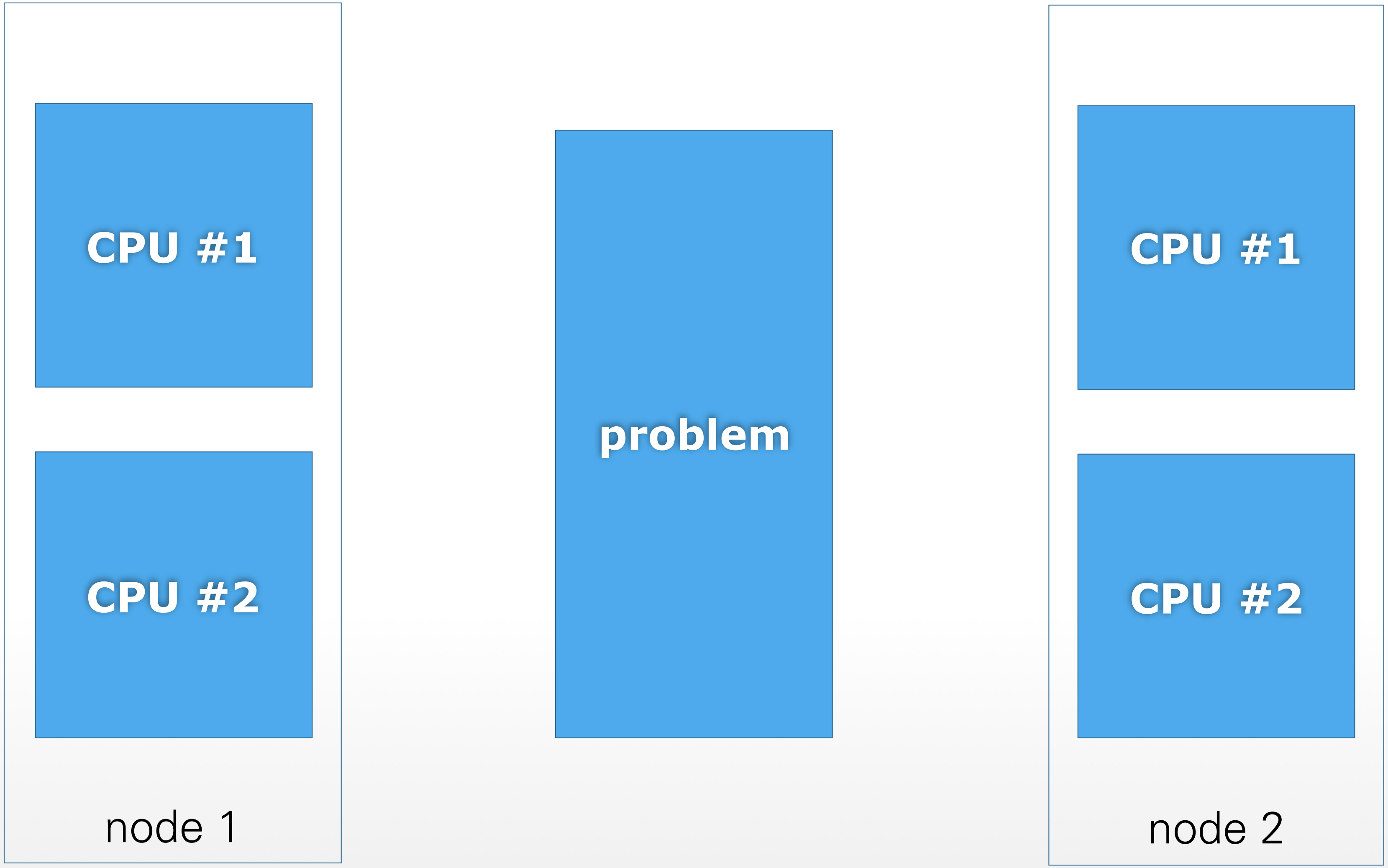
Not used for:

- Transaction-management system

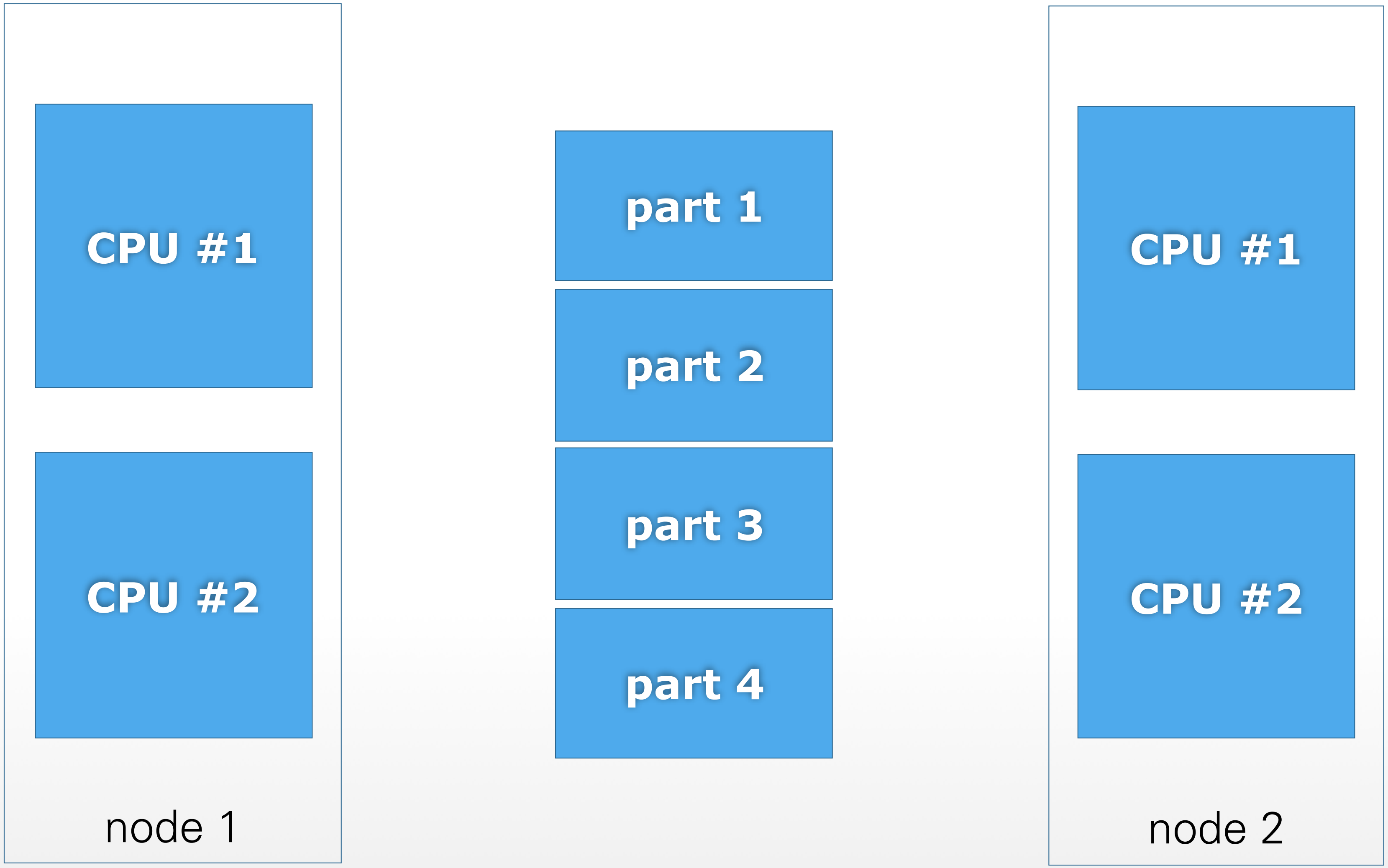
PROGRAMMING MODEL: SPMD

- Michael Flynn (1966):
  SISD, SIMD, MIMD, (MISD)SIMD
- SPMD: Single Program Multiple Data
  Same program runs on "all" nodes
  works on split-up data
  asynchronously but with explicit synch points
  implementations: message passing/shared memory/...paradigms: "map/reduce" (google) / GCD (apple) / task queues / ...
- often: while (true) { work; exchange data (barrier)}

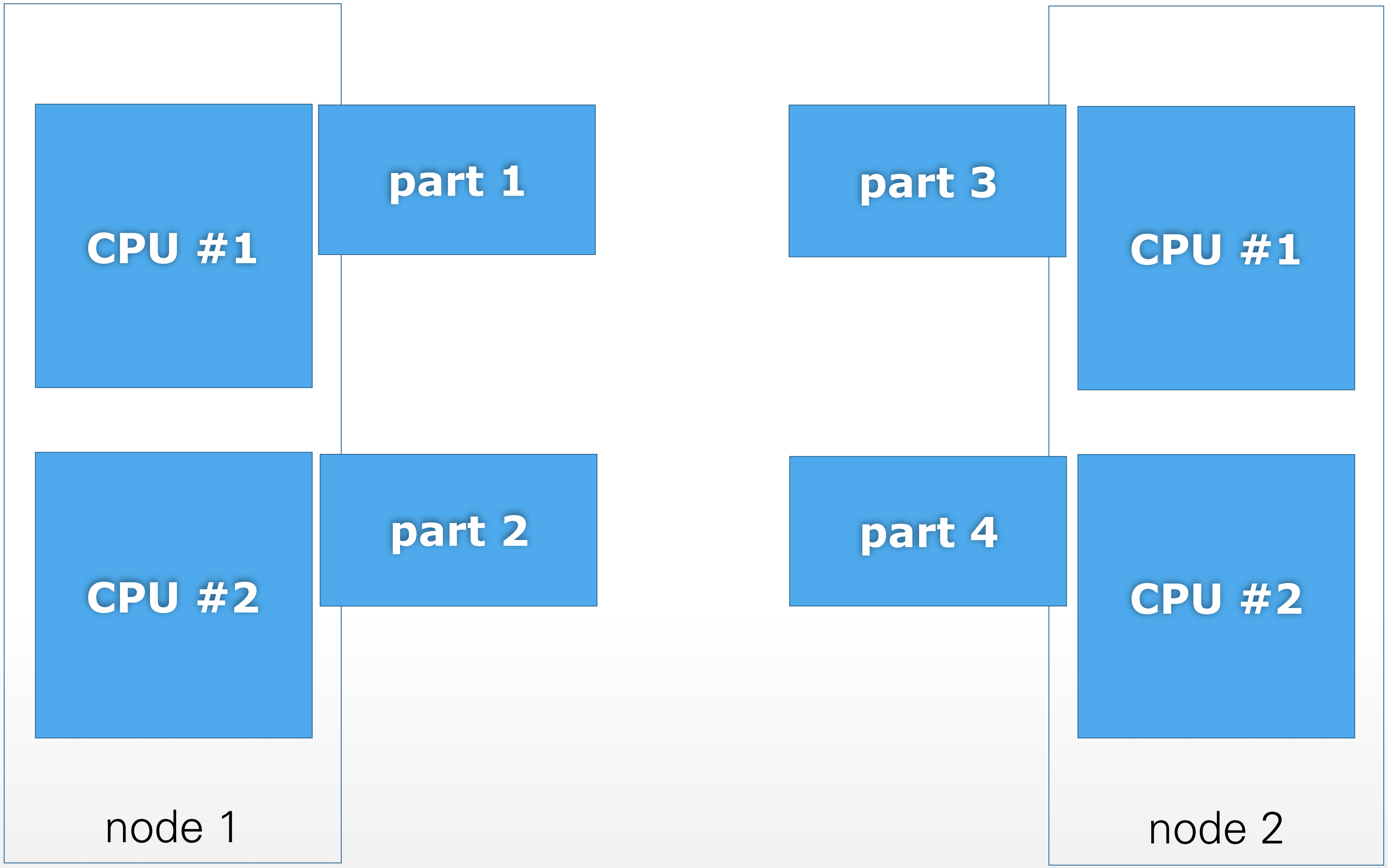
DIVIDE AND CONQUER



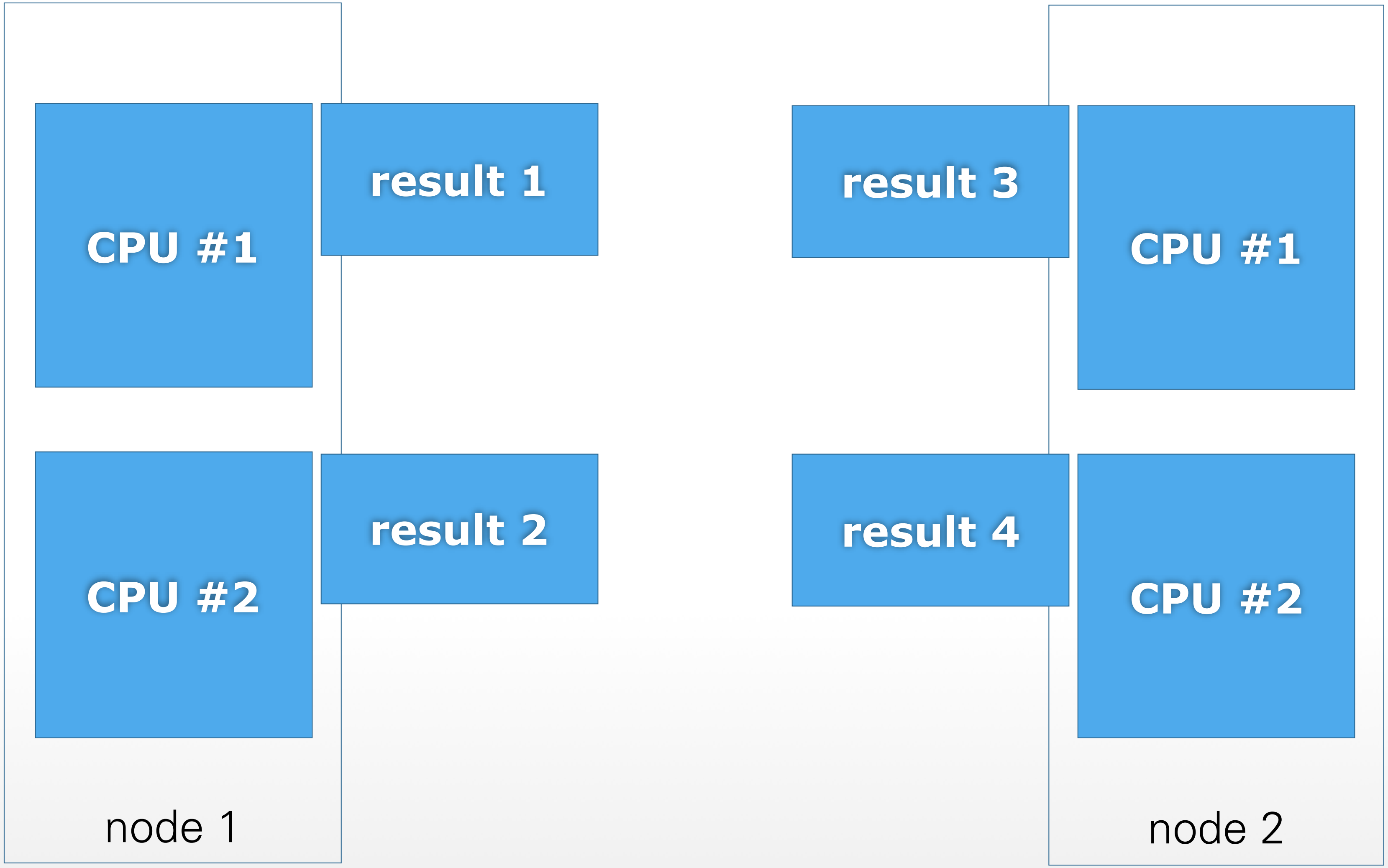
DIVIDE AND CONQUER



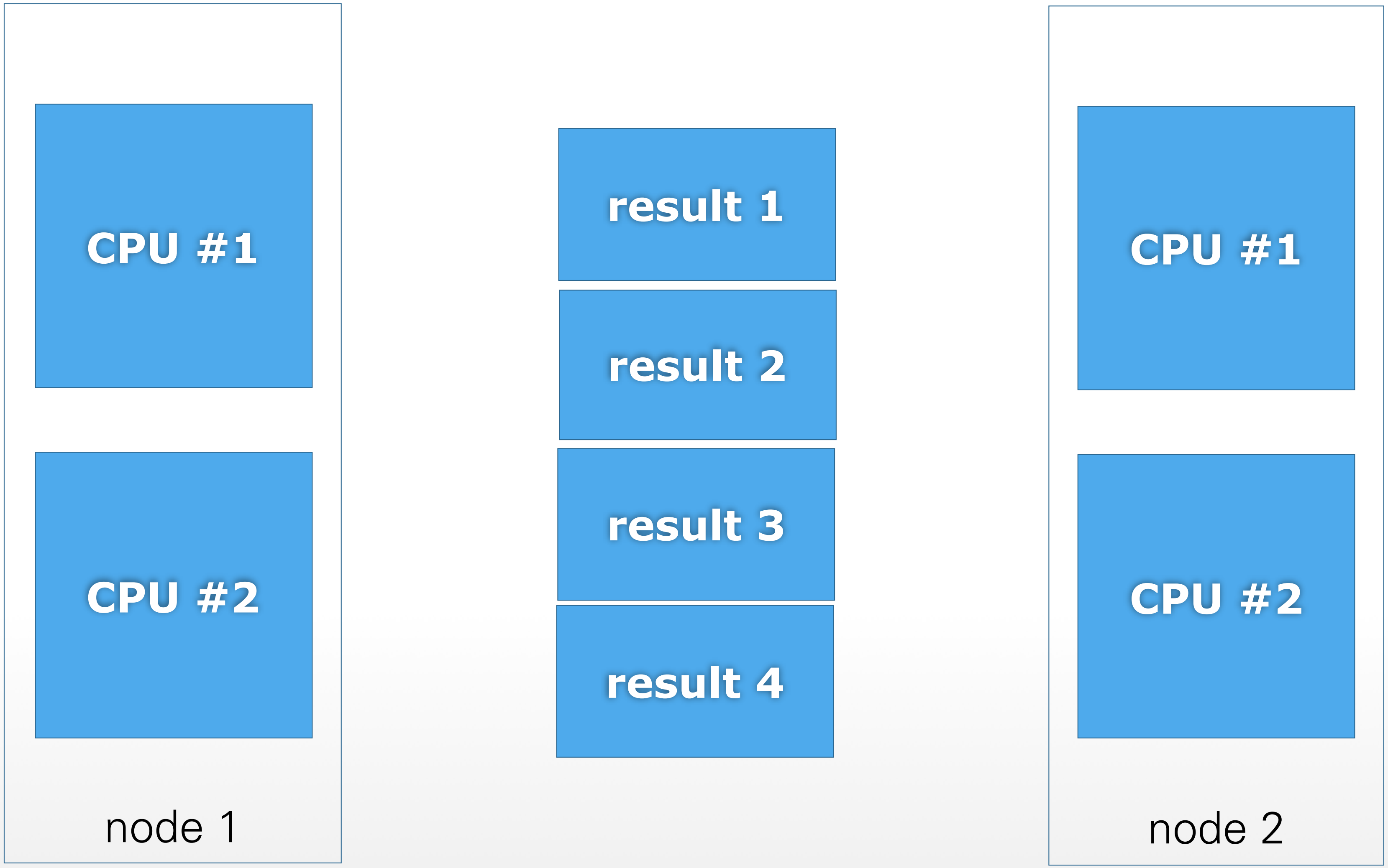
DIVIDE AND CONQUER



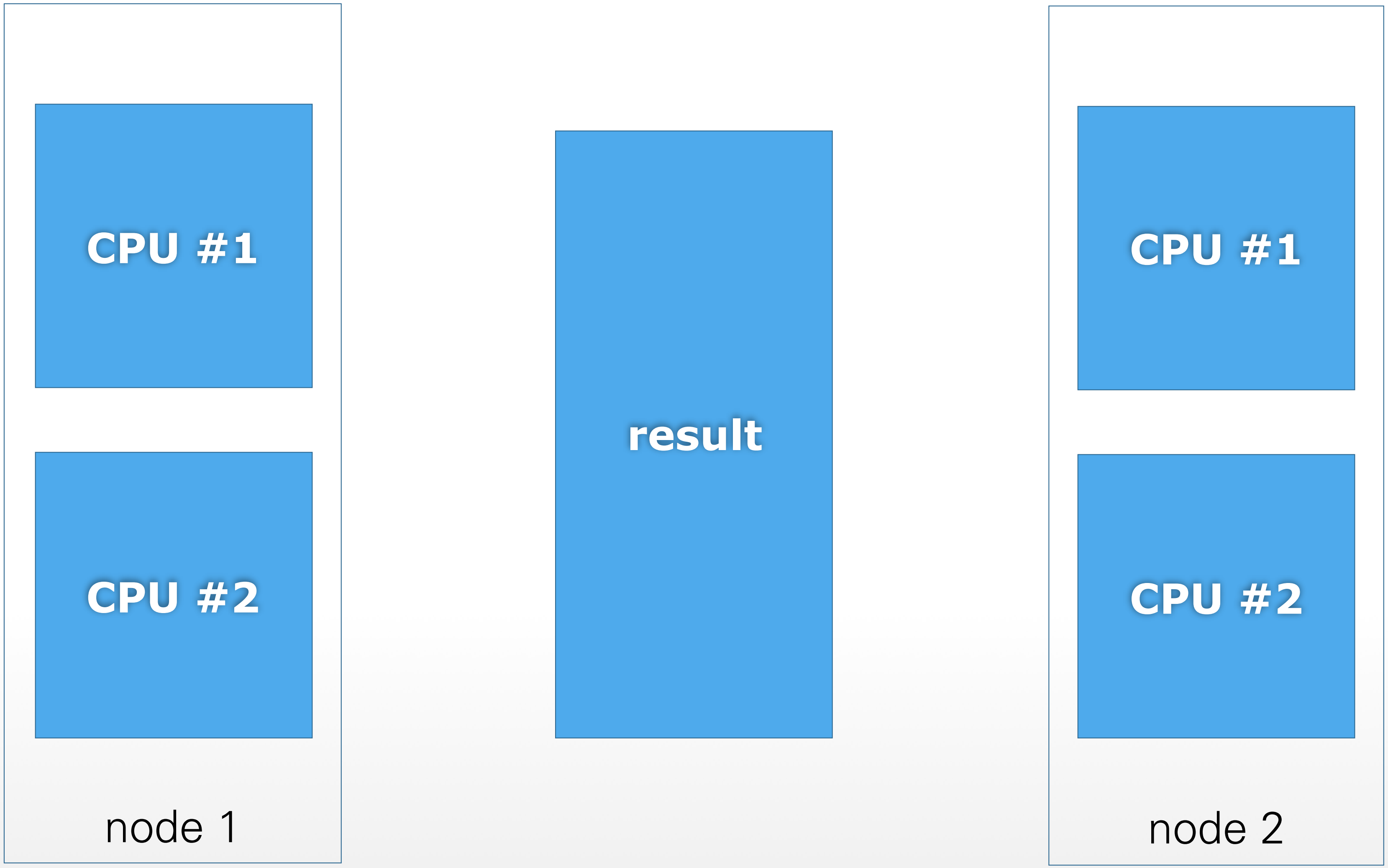
DIVIDE AND CONQUER



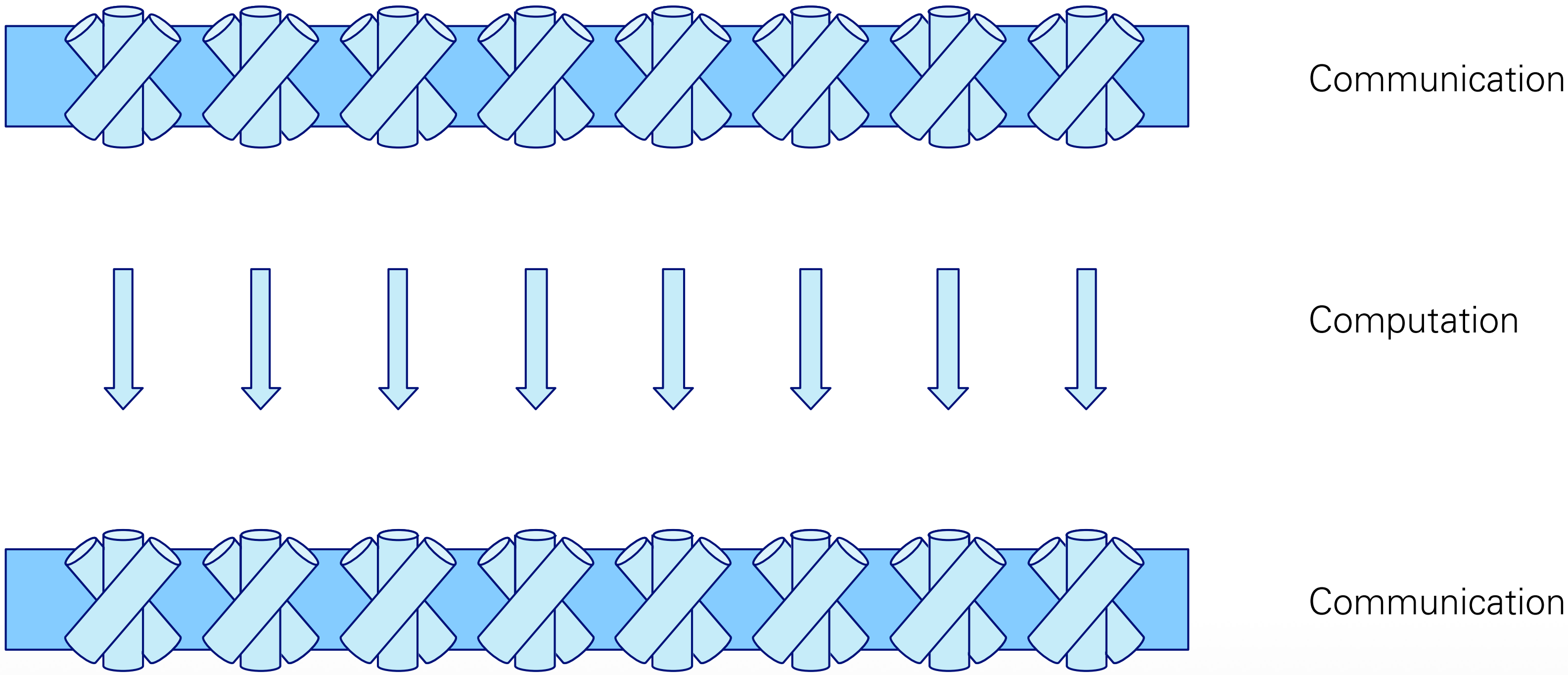
DIVIDE AND CONQUER



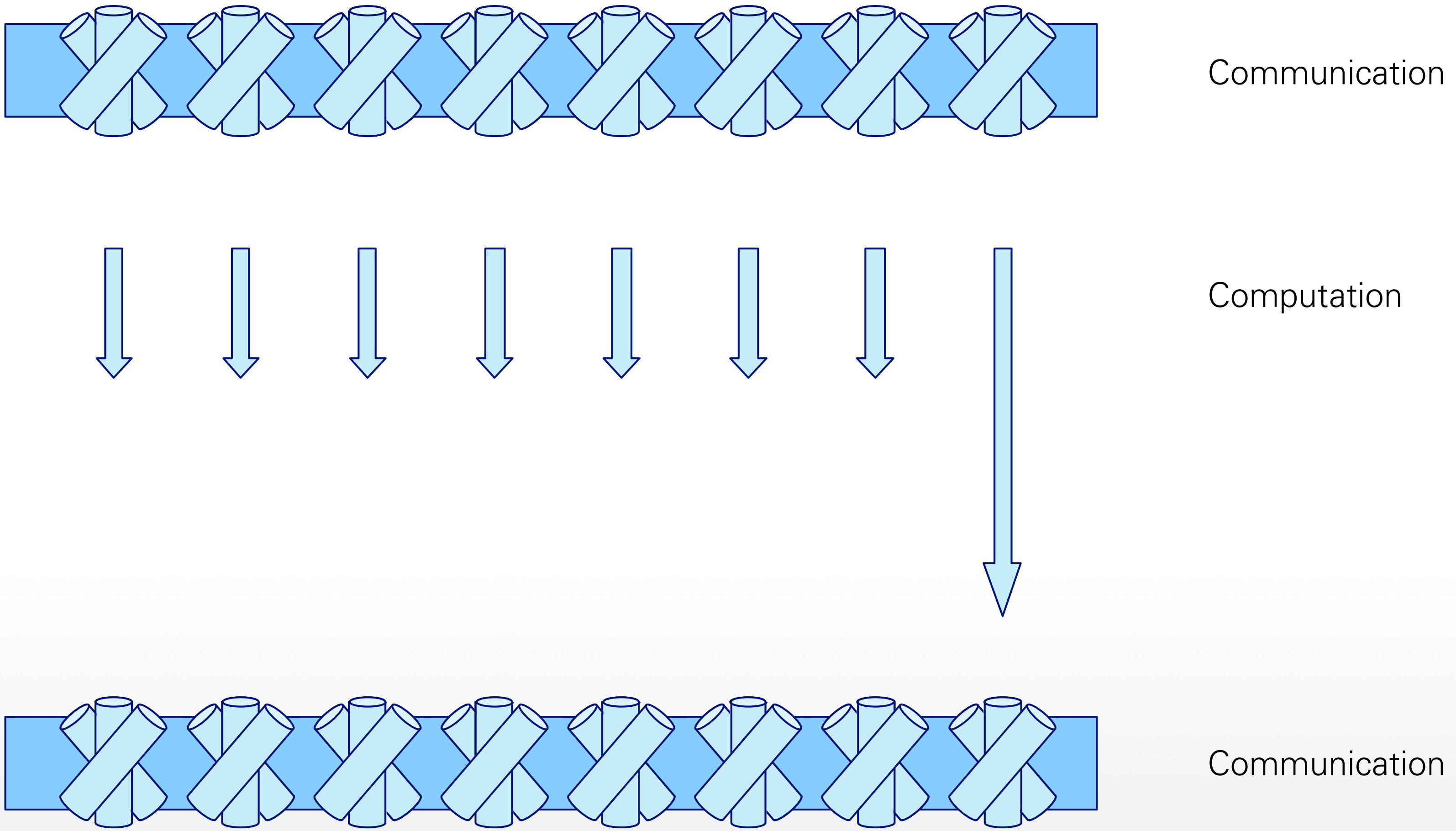
DIVIDE AND CONQUER



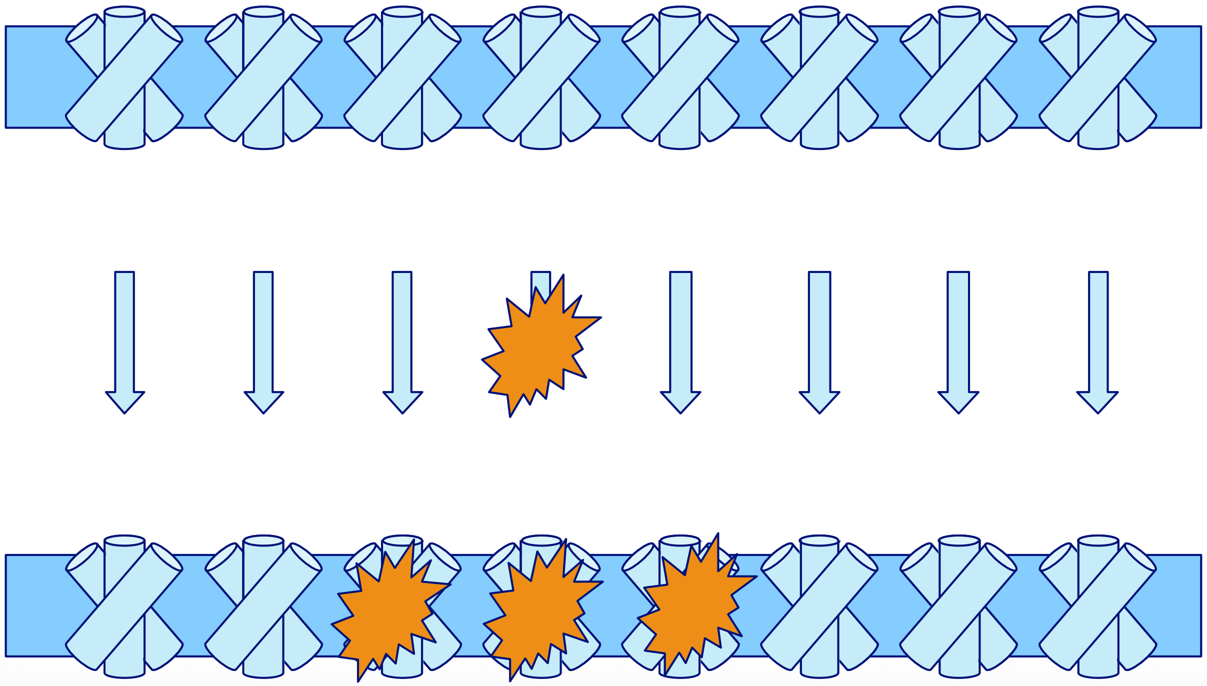
IMBALANCES & FAILURES



IMBALANCES & FAILURES



IMBALANCES & FAILURES



Communication

Computation

Communication

AMDAHL'S LAW

Compute; communicate; compute; …

- Examples (idealized, take with grain of salt !!!):
  - Compute: 10 micro, 100 micro, 1 ms
  - Communicate: 5 micro, 10 micro, 100 micro, 1ms
    assuming here: communication cannot be sped up

Amdahl's law: 1 / (1-P+P/N)

- P: section that can be parallelized
- 1-P: serial section
- N: number of CPUs

AMDAHL'S LAW

Compute(= parallel section),
communicate(= serial section)
→
possible speedup for N=∞

- 1ms, 100 µs: 1/0.1 → 10
- 1ms, 1 µs: 1/0.001 → 1000
- 10 µs, 1 µs: 0.01/0.001 → 10
- ...

TECHNISCHE UNIVERSITÄT DRESDEN

WEAK VS. STRONG SCALING

Strong:

- accelerate same problem size

Weak:

- extend to larger problem size

AMDAHL'S LAW

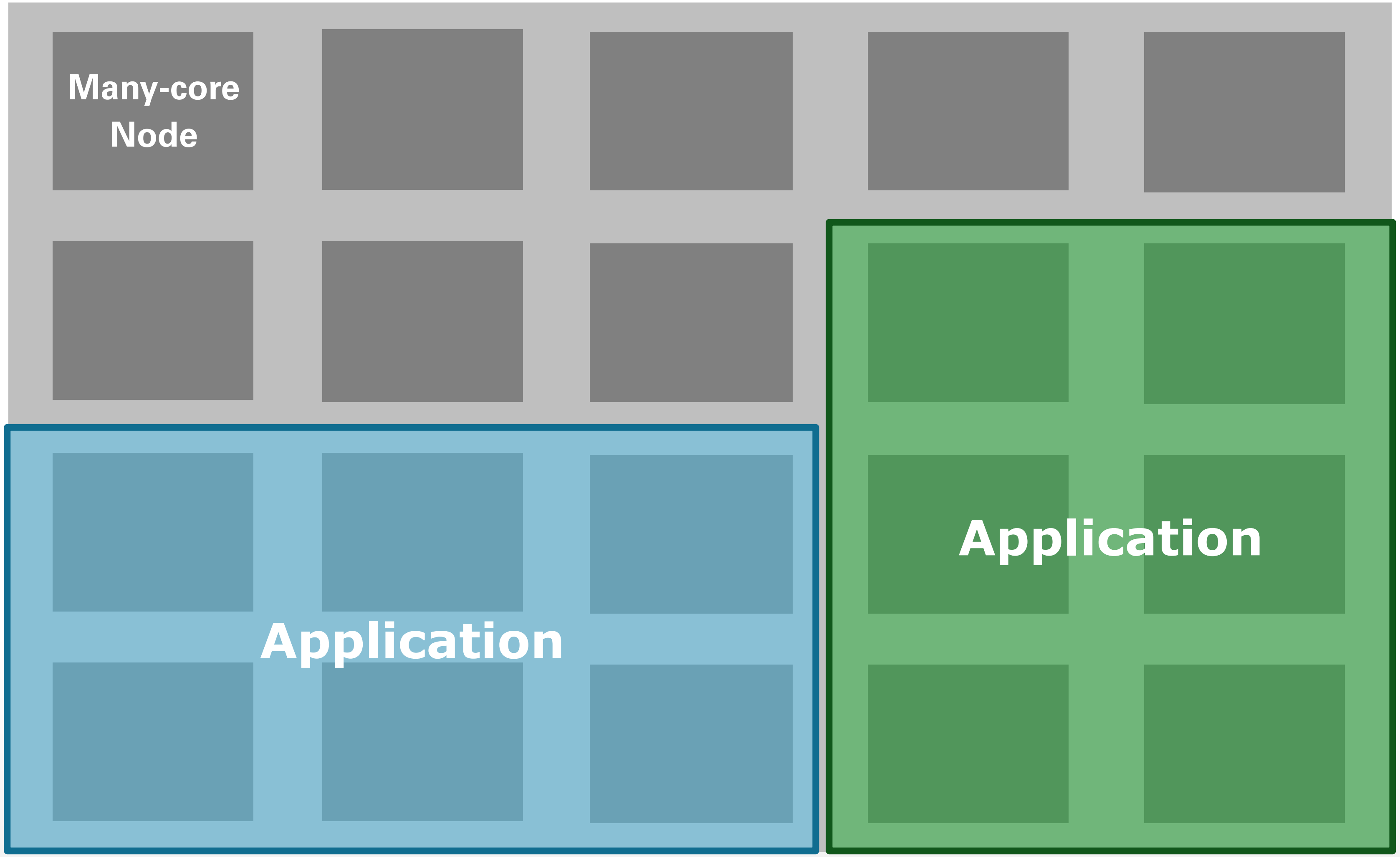
Jitter, "Noise", "micro scrabblers":

- Occasional addition to computation/communication time in one or more processes
- Holds up all other processes

Compute(= parallel section),
jitter (→ add to serial section),
communicate(= serial section):
possible speedup for N=∞

■ 1ms, 100µs, 100 µs:	1/0.2	→ 5	(10)
■ 1ms, 100µs, 1 µs:	1/0.101	→ 10	(1000)
■ 10 µs, 10µs, 1 µs:	0.01/0.011	→ 1	(10)

STATE OF THE ART IN HPC



STATE OF THE ART IN HPC

- dedicate full partition to application
  (variant: "gang scheduling")
- load balancing done (tried) by applications or user-level runtime (Charm++)
- avoid OS calls
- "scheduler":
  manages queue of application processes
  assigns partitions to applications
  supervises run-time
- applications run from checkpoint to checkpoint

TECHNISCHE UNIVERSITÄT DRESDEN

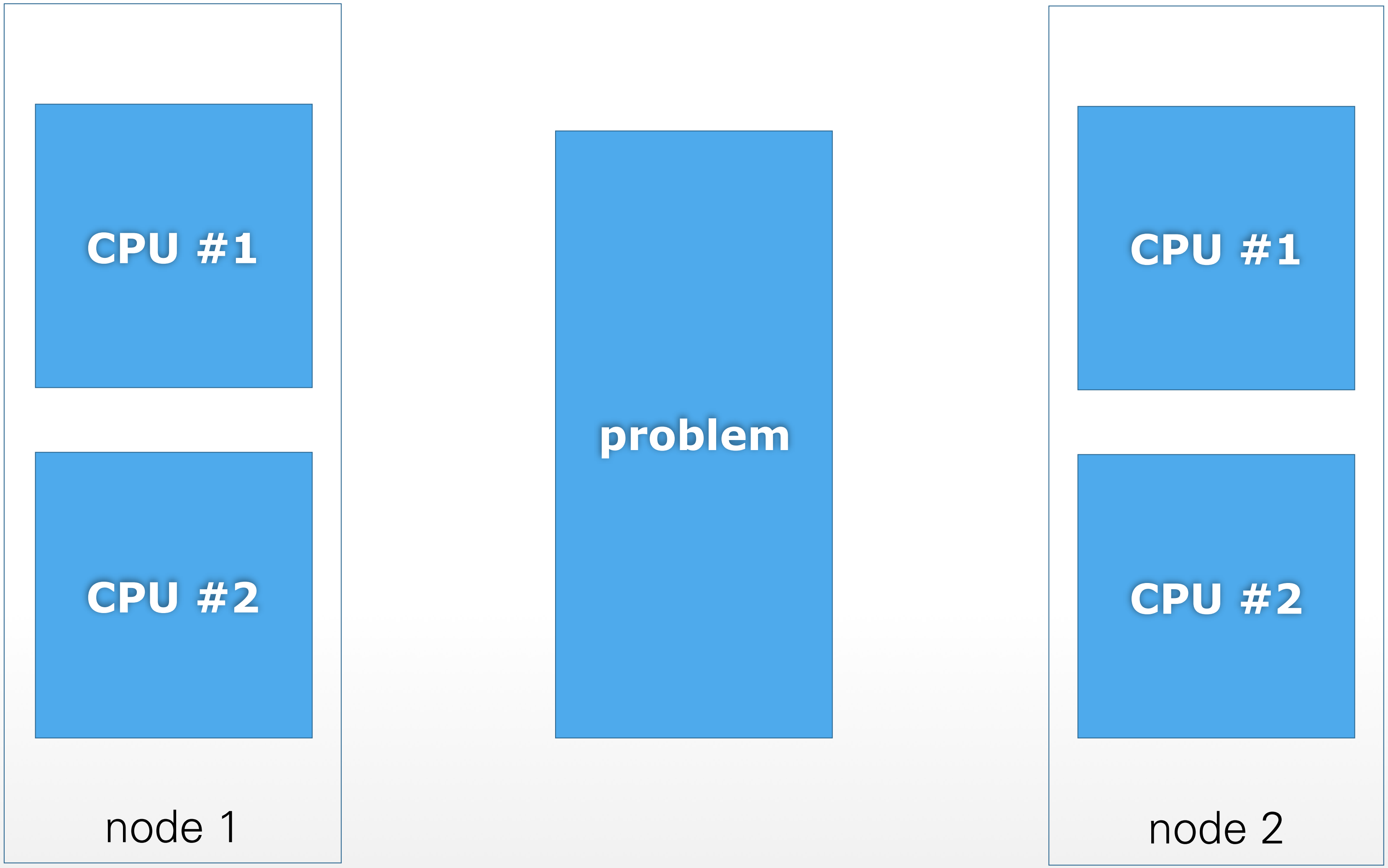
STATE OF THE ART IN HPC: RDMA

- nodes access remote memory via load/store operations
- busy waiting across nodes (within partition)
- barrier ops supported by network
- compare&exchange on remote memory operation
- no OS calls for message ops (busy waiting)

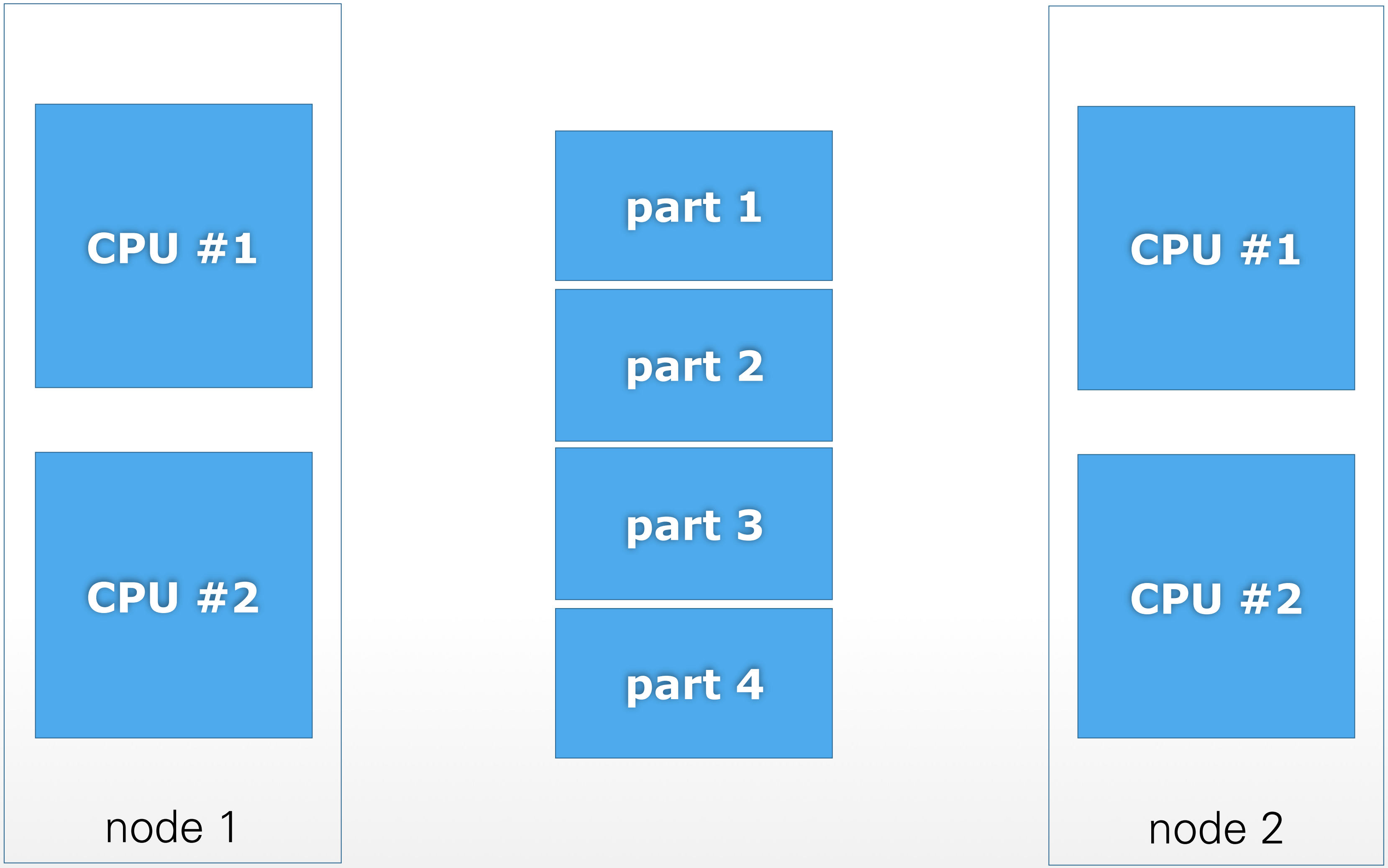
MPI BRIEF OVERVIEW

- Library for message-oriented parallel programming
- Programming model:
  - Multiple instances of same program
  - Independent calculation
  - Communication, synchronization

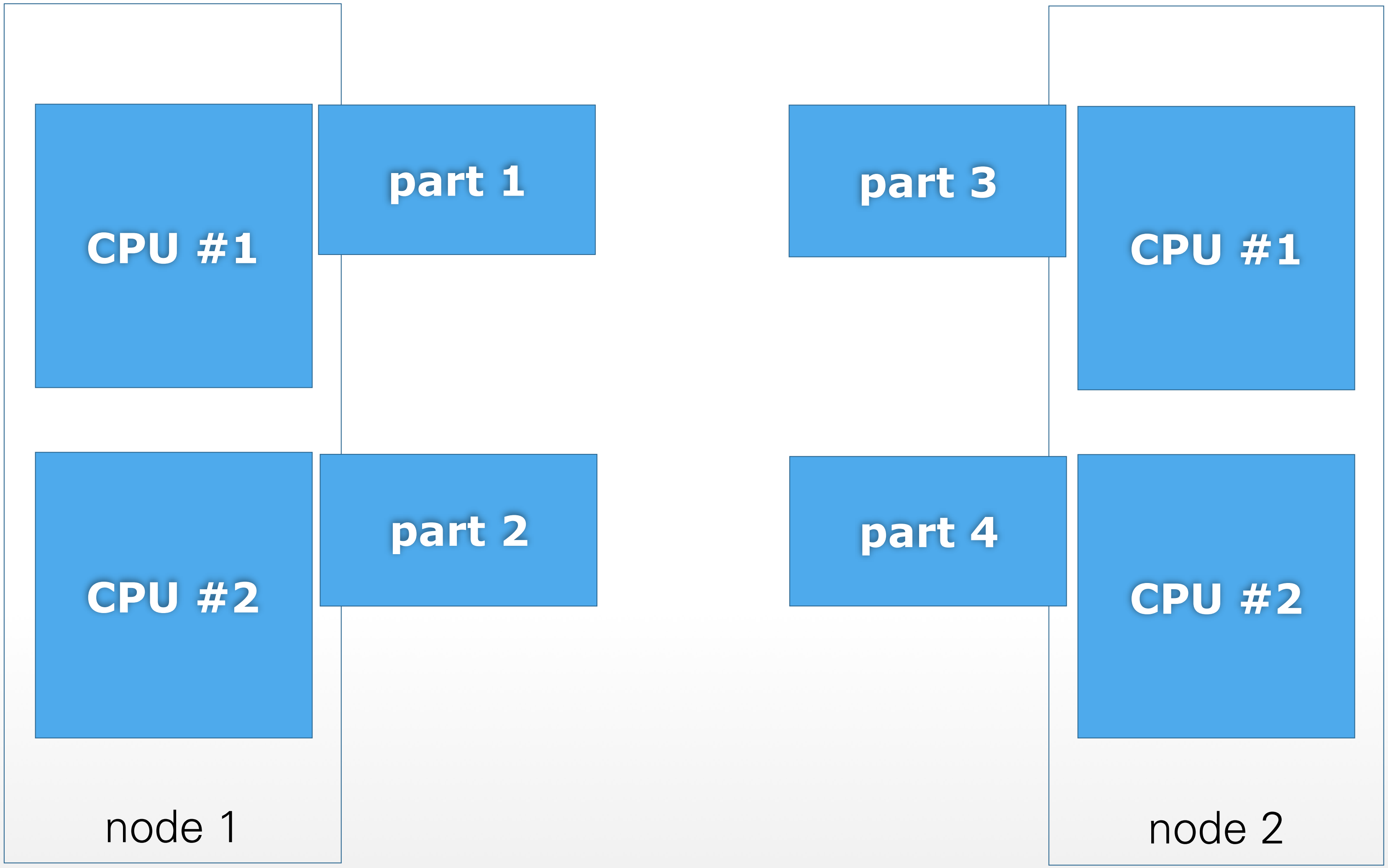
DIVIDE AND CONQUER



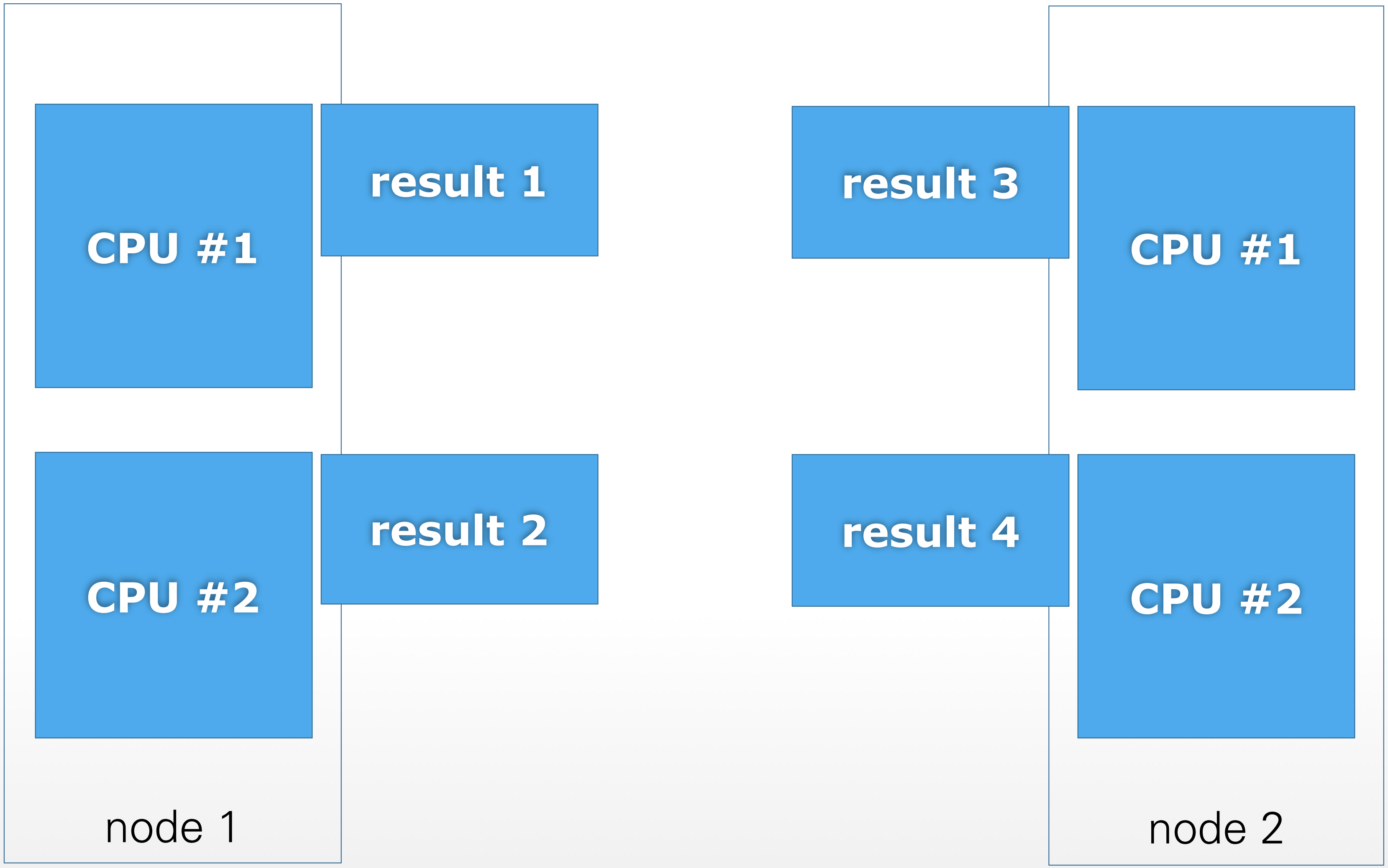
DIVIDE AND CONQUER



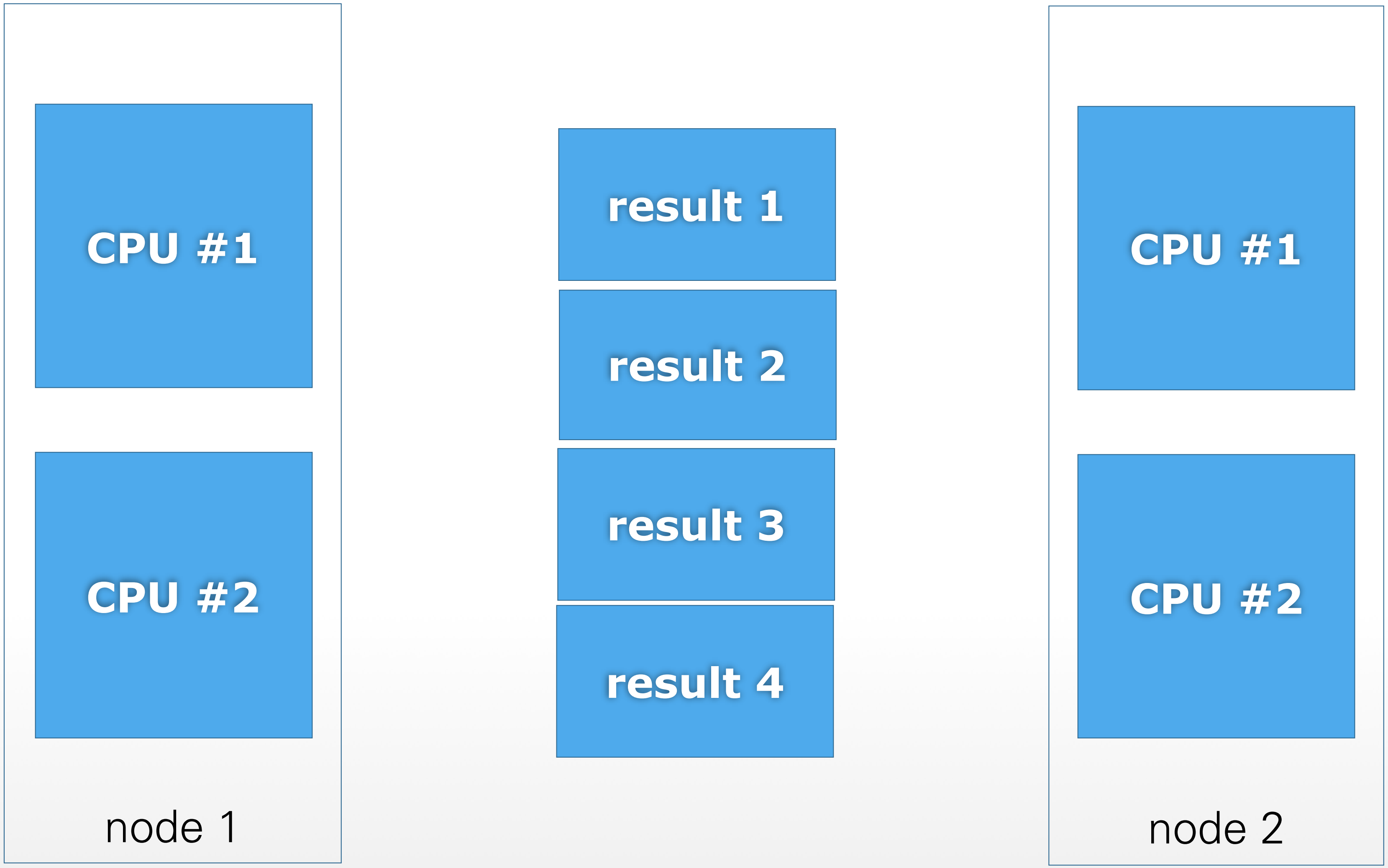
DIVIDE AND CONQUER



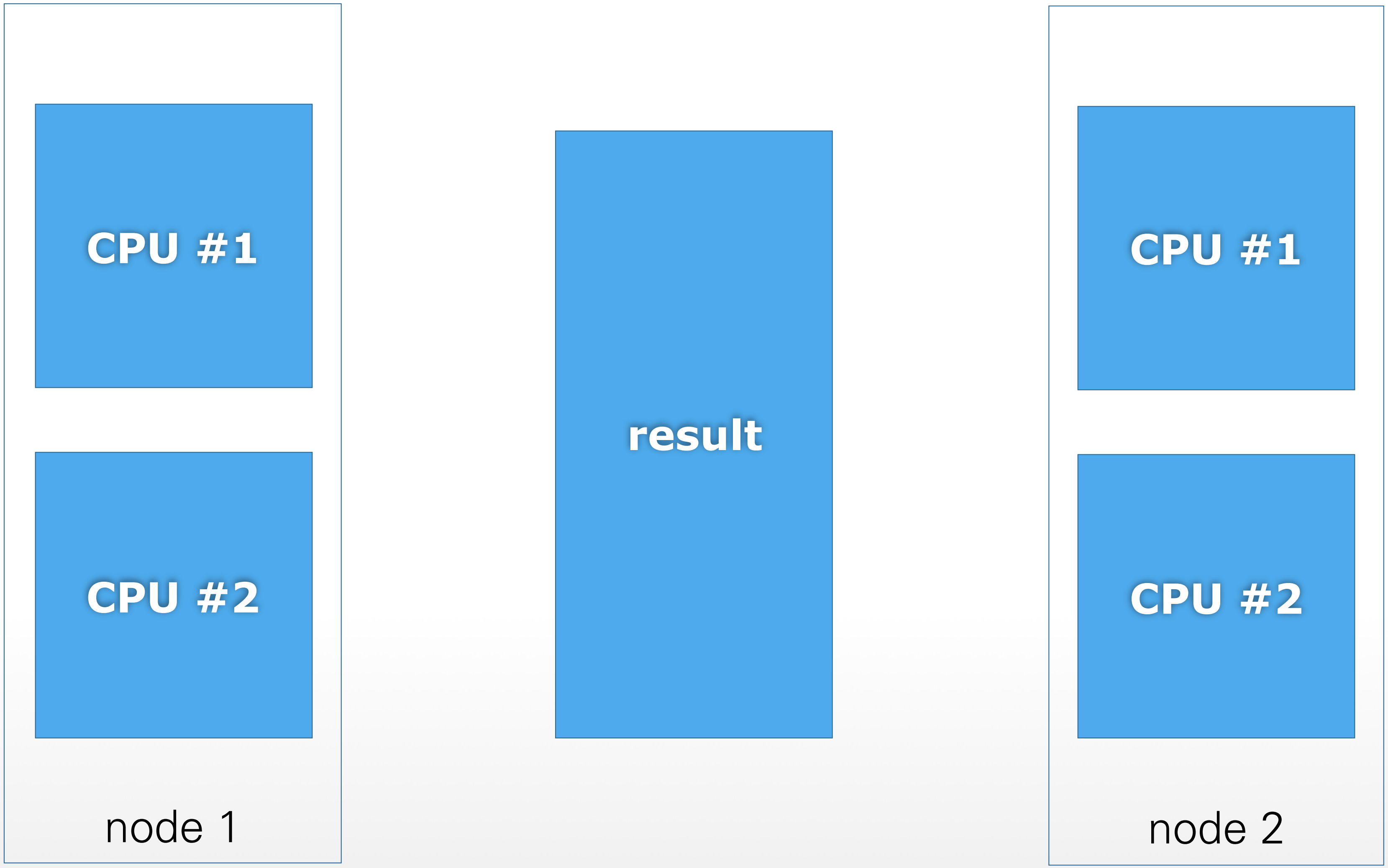
DIVIDE AND CONQUER



DIVIDE AND CONQUER



DIVIDE AND CONQUER



MPI STARTUP & TEARDOWN

- MPI program is started on all processors
- `MPI_Init()`, `MPI_Finalize()`
- Communicators (e.g., MPI_COMM_WORLD)
  - `MPI_Comm_size()`
  - `MPI_Comm_rank()`: "Rank" of process within this set
  - Typed messages
- Dynamically create and spread processes using `MPI_Spawn()` (since MPI-2)

MPI EXECUTION

MPI EXECUTION

- Communication

MPI EXECUTION

- Communication
  - Point-to-point

```
MPI_Send(
  void* buf,
  int count,
  MPI_Datatype,
  int dest,
  int tag,
  MPI_Comm comm
)
```

MPI EXECUTION

- Communication
  - Point-to-point

```
MPI_Recv(
   void* buf,
   int count,
   MPI_Datatype,
   int source,
   int tag,
   MPI_Comm comm,
   MPI_Status *status
)
```

MPI EXECUTION

- Communication
  - Point-to-point
  - Collectives

```
MPI_Bcast(
   void* buffer,
   int count,
   MPI_Datatype,
   int root,
   MPI_Comm comm
)
```

MPI EXECUTION

- Communication
  - Point-to-point
  - Collectives

```
MPI_Reduce(
   void* sendbuf,
   void *recvbuf,
   int count
   MPI_Datatype,
   MPI_Op op,
   int root,
   MPI_Comm comm
)
```

MPI EXECUTION

- Communication
  - Point-to-point
  - Collectives
- Synchronization

MPI EXECUTION

- Communication
  - Point-to-point
  - Collectives
- Synchronization
  - Test

```
MPI_Test(
   MPI_Request* request,
   int *flag,
   MPI_Status *status
)
```

MPI EXECUTION

- Communication
  - Point-to-point
  - Collectives
- Synchronization
  - Test
  - Wait

```
MPI_Wait(
  MPI_Request* request,
  MPI_Status *status
)
```

MPI EXECUTION

- Communication
  - Point-to-point
  - Collectives
- Synchronization
  - Test
  - Wait
  - Barrier

```
MPI_Barrier(
 MPI_Comm comm
)
```

BLOCK AND SYNC

	blocking call	non-blocking call
synchronous communication		
asynchronous communication		

BLOCK AND SYNC

	blocking call	non-blocking call
synchronous communication	returns when message has been delivered	
asynchronous communication		

BLOCK AND SYNC

	blocking call	non-blocking call
synchronous communication	returns when message has been delivered	
asynchronous communication	returns when send buffer can be reused	

BLOCK AND SYNC

	blocking call	non-blocking call
synchronous communication	returns when message has been delivered	returns immediately, following test/wait checks for delivery
asynchronous communication	returns when send buffer can be reused	

BLOCK AND SYNC

	blocking call	non-blocking call
synchronous communication	returns when message has been delivered	returns immediately, following test/wait checks for delivery
asynchronous communication	returns when send buffer can be reused	returns immediately, following test/wait checks for send buffer

EXAMPLE

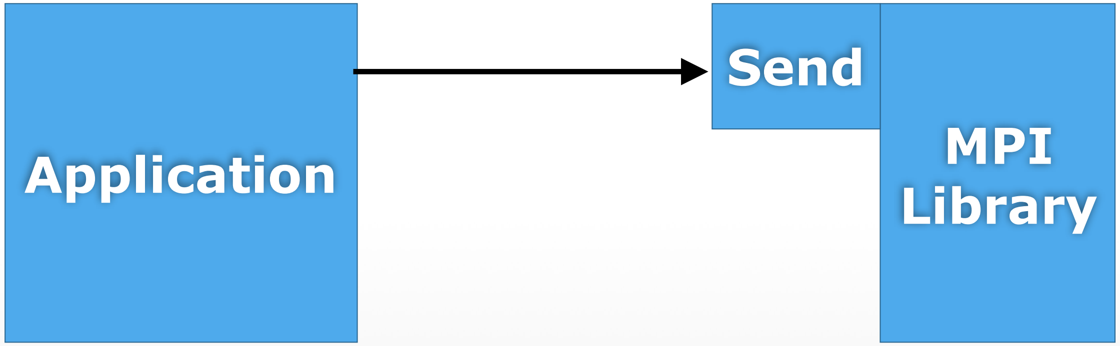
```
int rank, total;
MPI_Init();
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &total);

MPI_Bcast(...);
/* work on own part, determined by rank */

if (id == 0) {
  for (int rr = 1; rr < total; ++rr)
    MPI_Recv(...);
  /* Generate final result */
} else {
    MPI_Send(...);
}
MPI_Finalize();
```

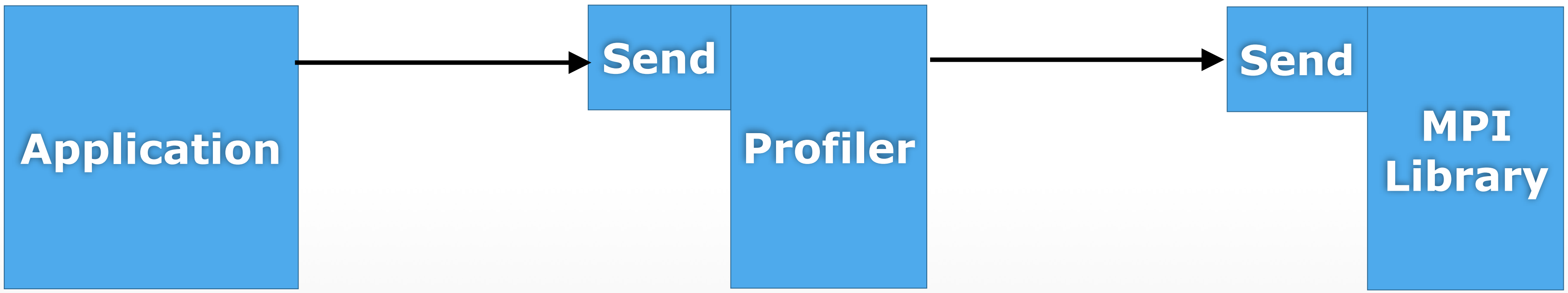
PMPI

- Interposition layer between library and application
- Originally designed for profiling



PMPI

- Interposition layer between library and application
- Originally designed for profiling



EXA-SCALE: HW+SW ASSUMPTIONS

- Large number of nodes:
  - Many compute cores
  - 1 or 2 service cores
- Failure rate exceeds checkpoint rate
- Fast local persistent storage on each node
- Not all cores available all the time (dark silicon due to heat/energy issues)
- Compute + communication heavy applications, may not be balanced
- short term changes of frequency ?

ROLE OF OPERATING SYSTEM

- for applications with extreme (bad) computation/ communications ratio:
  NOT MUCH, but
  -> avoid "noise", use common sense
- all others:
  handle faults
  use dark silicon
  balance load
  gossip
  over decomposition & over subscription
  predict execution times
  use scheduling tricks
  optimise for network/memory topology

OPERATING SYSTEM "NOISE"

Use common sense to avoid:

- OS usually not directly on the critical path,
  BUT OS controls: interference via interrupts, caches, network, memory bus, (RTS techniques)
- avoid or encapsulate side activities
- small critical sections (if any)
- partition networks to isolate traffic of different applications (HW: Blue Gene)
- do not run Python scripts or printer daemons in parallel

FFMK@TU-DRESDEN +

+ Hebrew Uni (Mosix team) + ZIB (FS team)
Fast and Fault-Tolerant Microkernel-based OS

- get rid of partitions
- use a micro-kernel (L4)
- OS supported load balancing
- use RAID for fast checkpoints

DFG-supported

4 TECHNOLOGIES

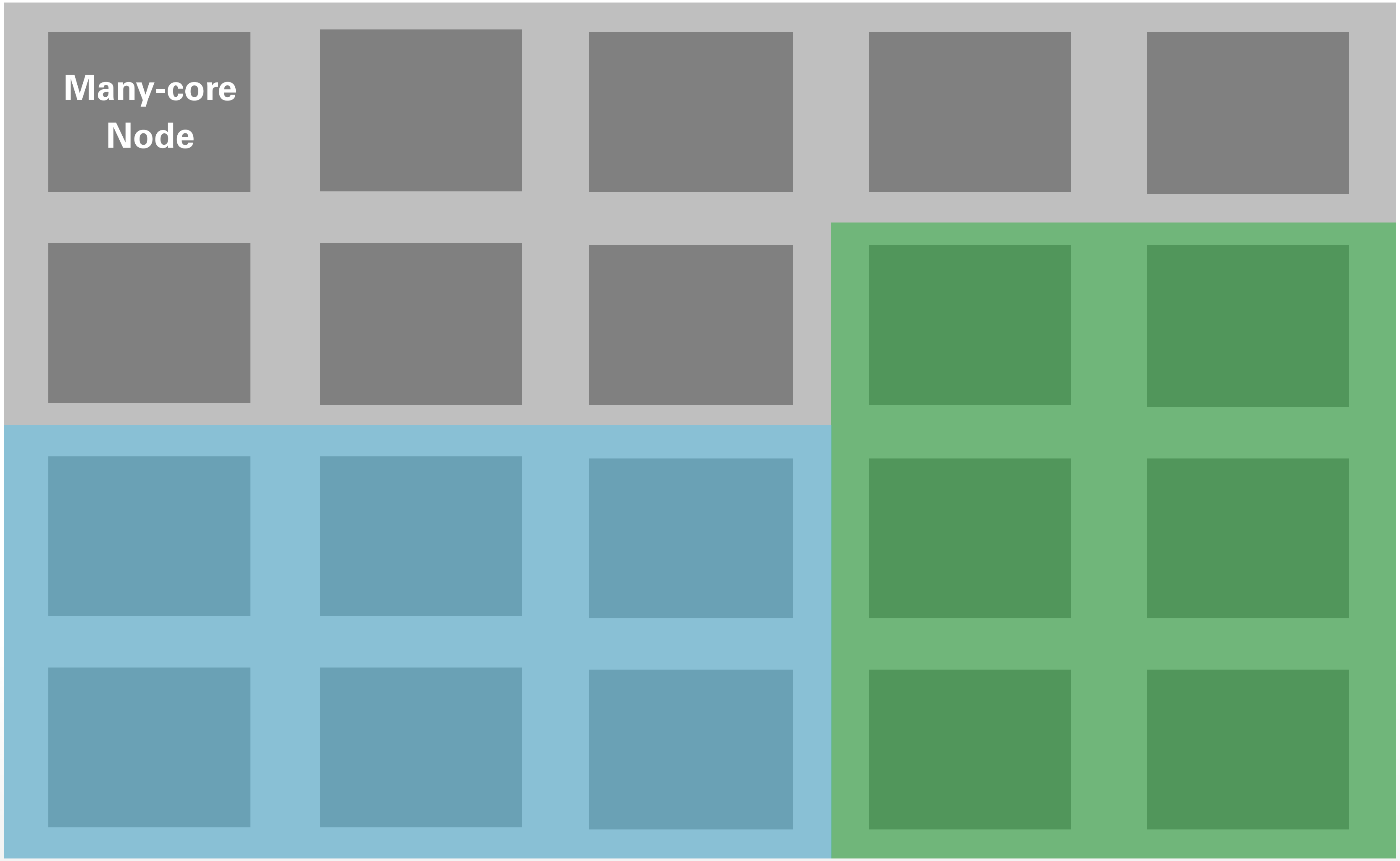
Microkernels, virtualization, split architectures

MOSIX-style online system management (gossip)

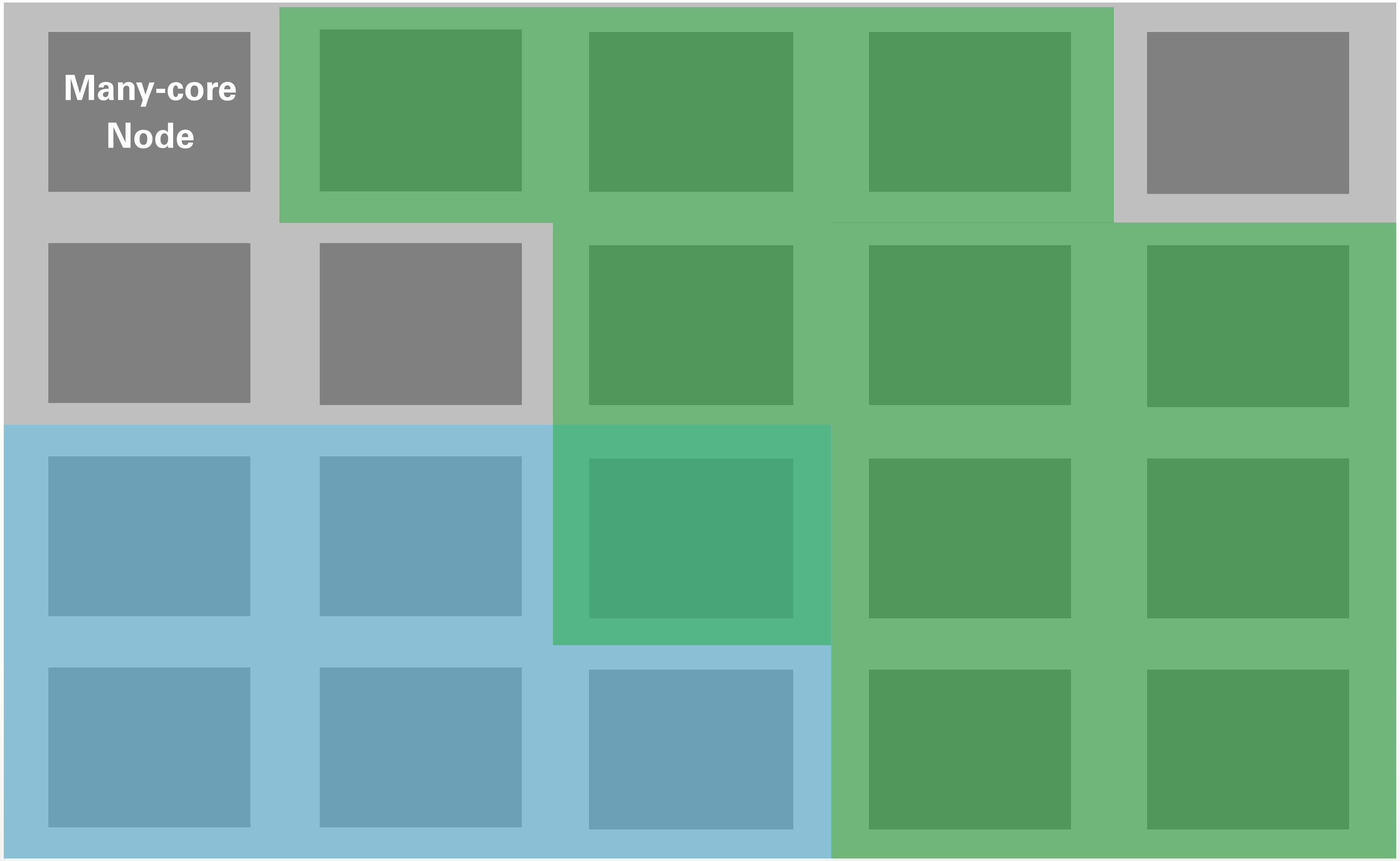
Distributed in-memory (on-node) checkpointing

MPI + applications

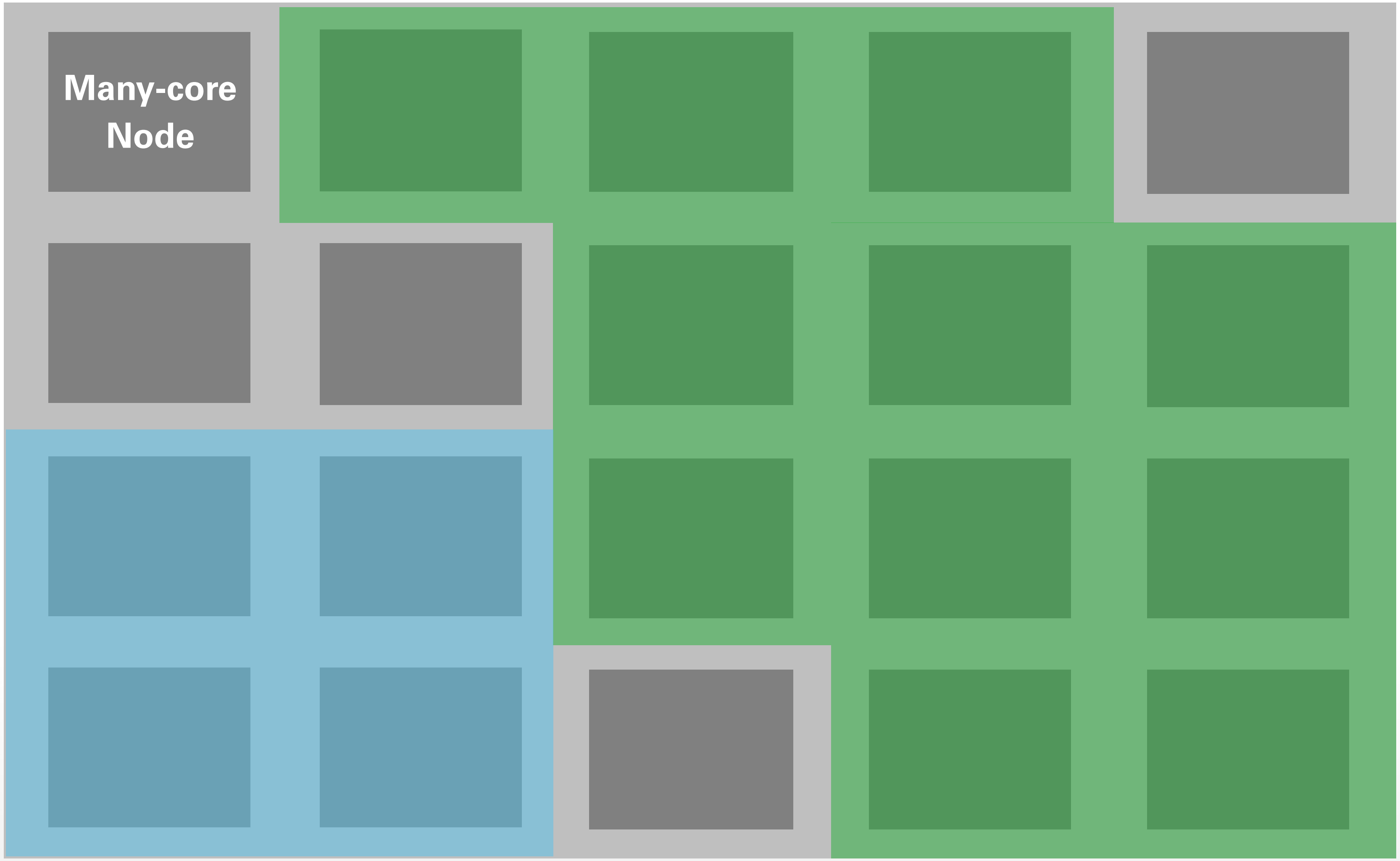
GOAL FOR EXASCALE HPC



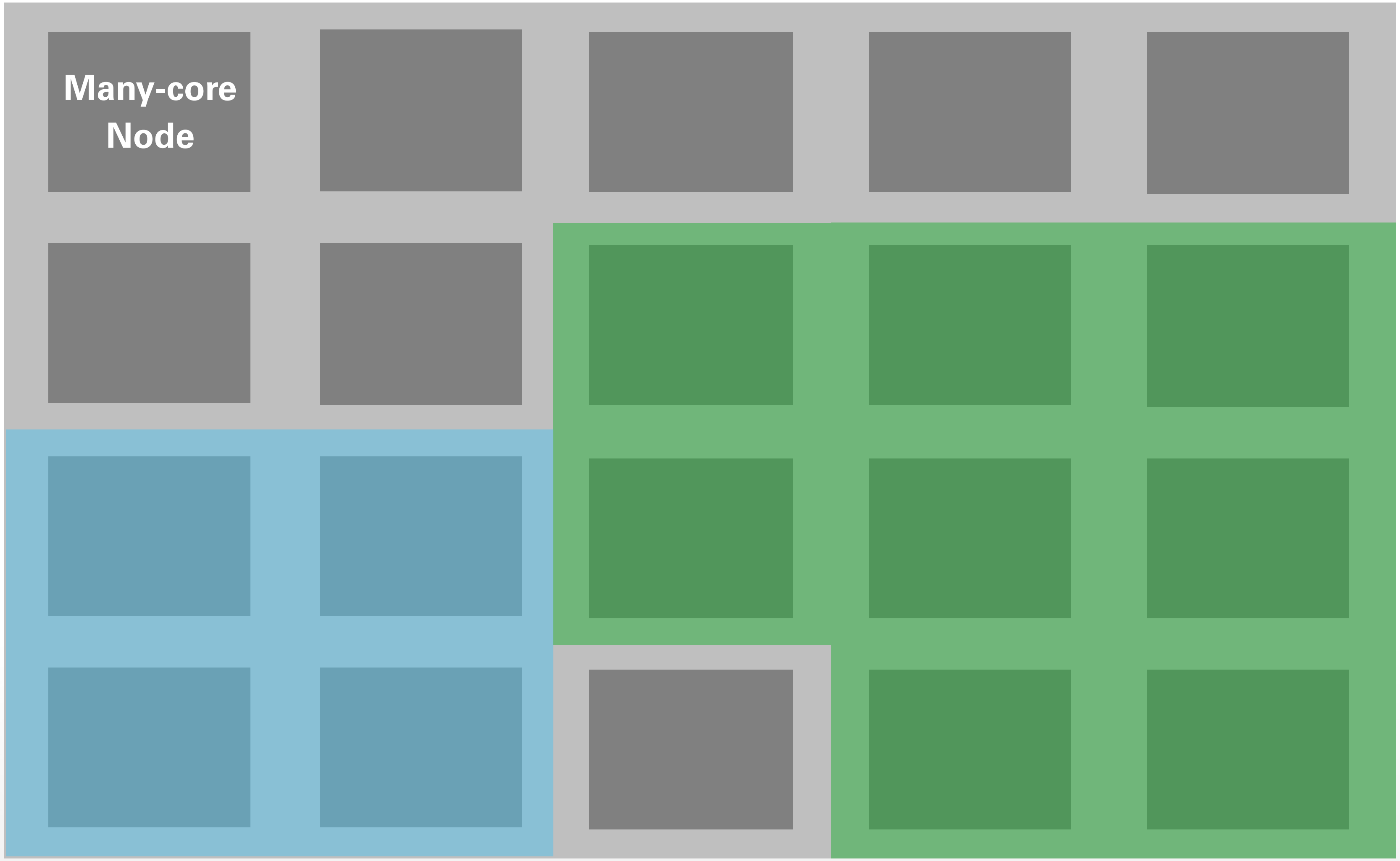
GOAL FOR EXASCALE HPC



GOAL FOR EXASCALE HPC



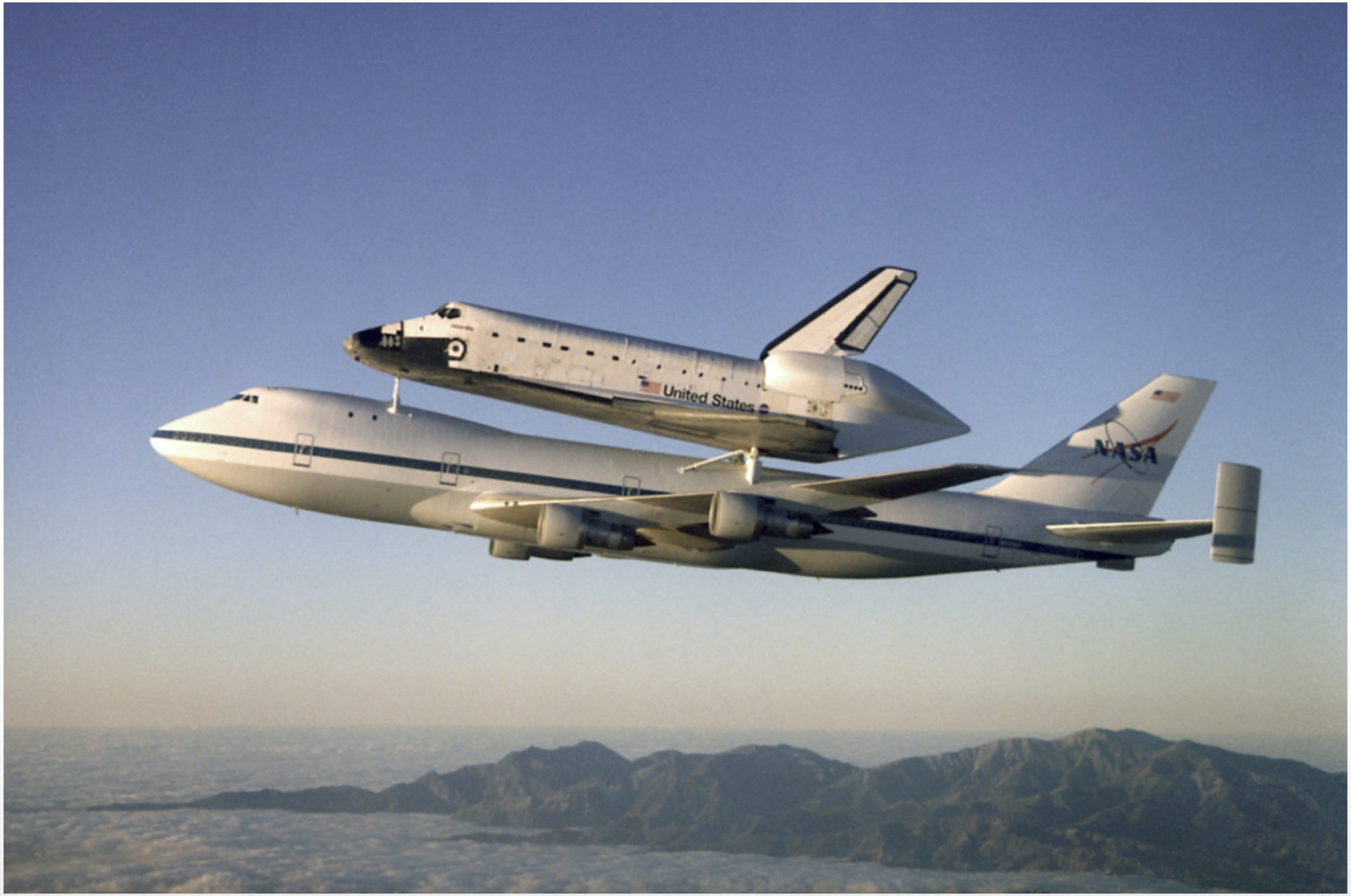
GOAL FOR EXASCALE HPC



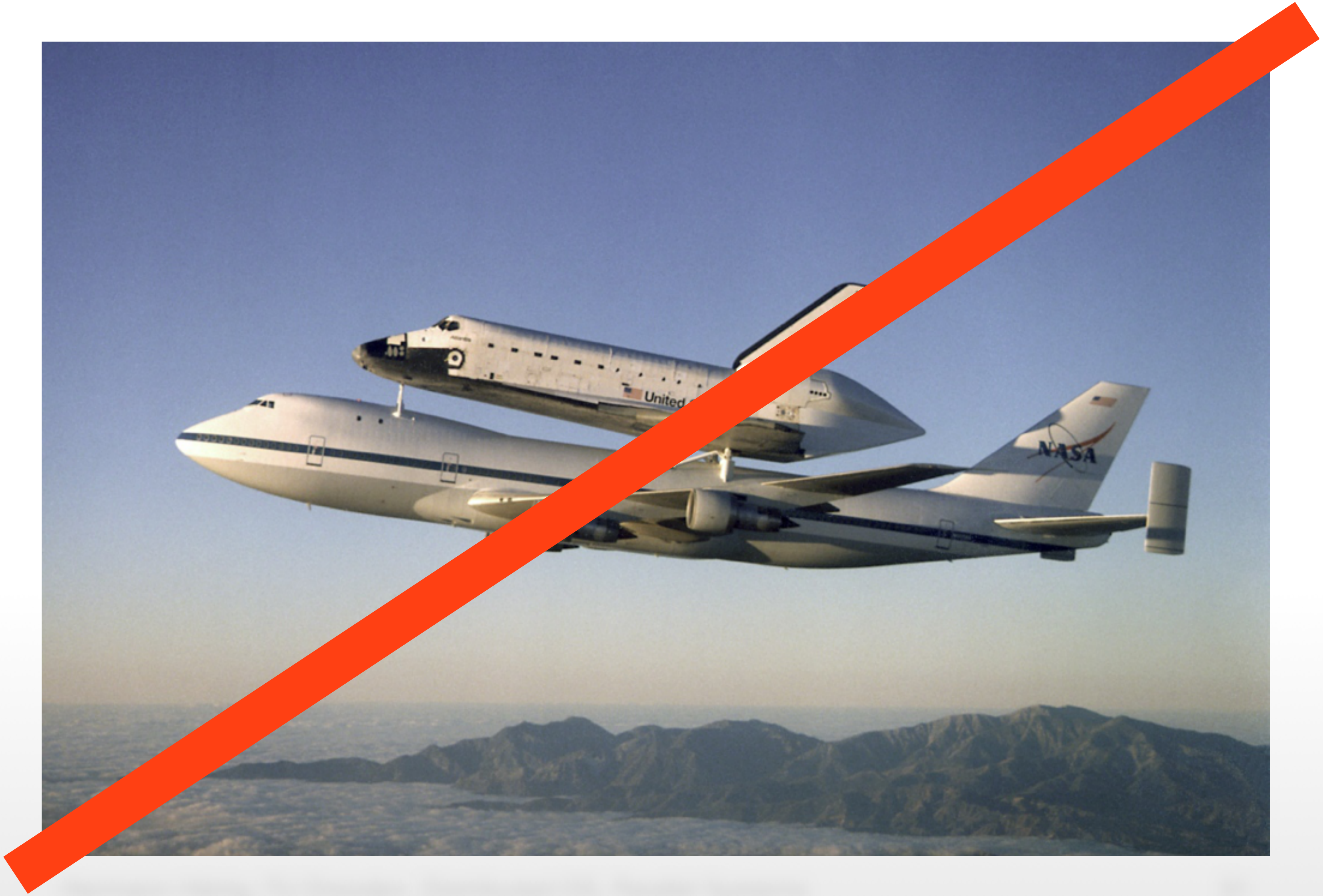
TECHNISCHE UNIVERSITÄT DRESDEN

THIN COMMON SUBSTRATE

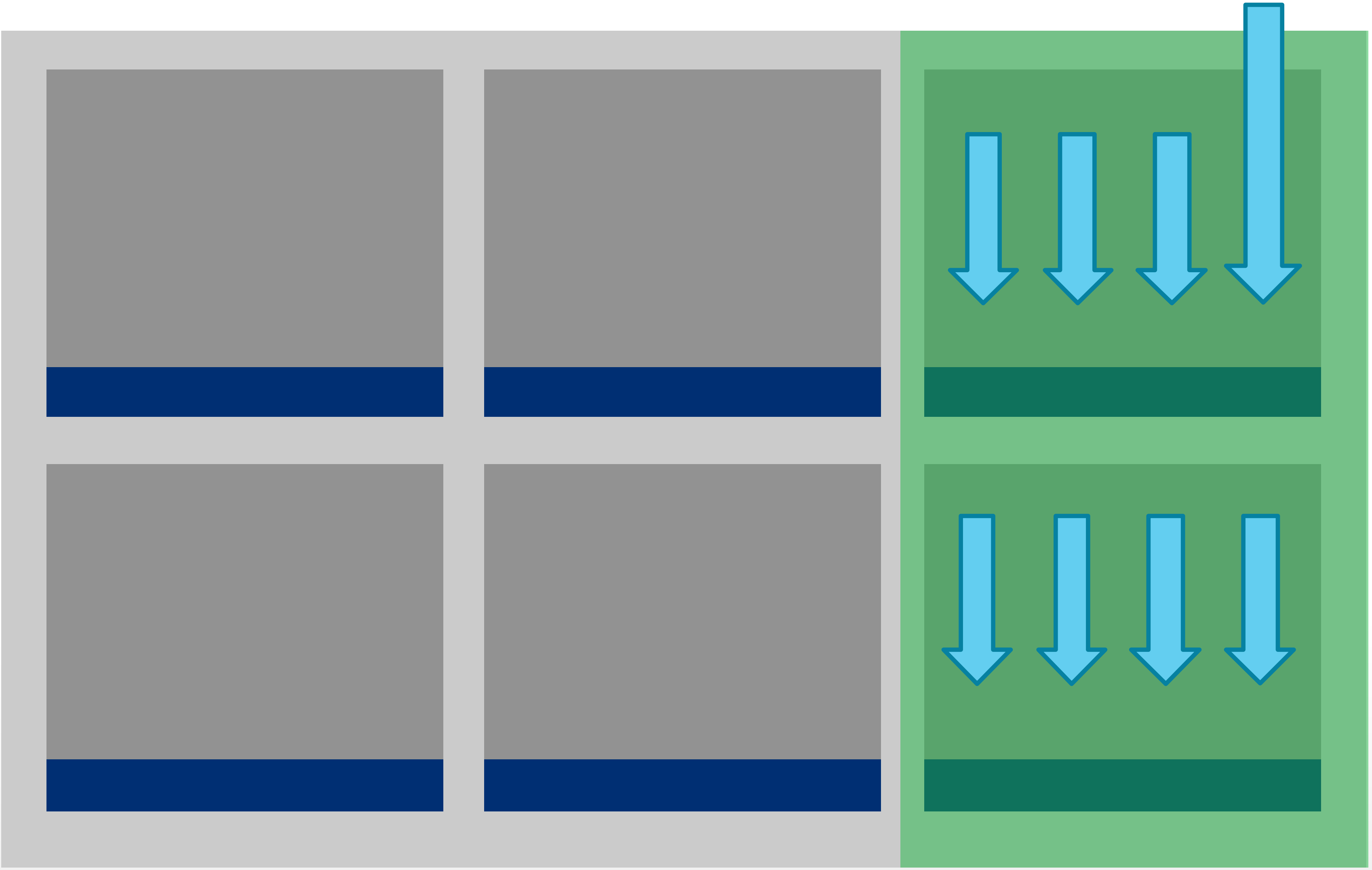
SMALL? PREDICTABLE?



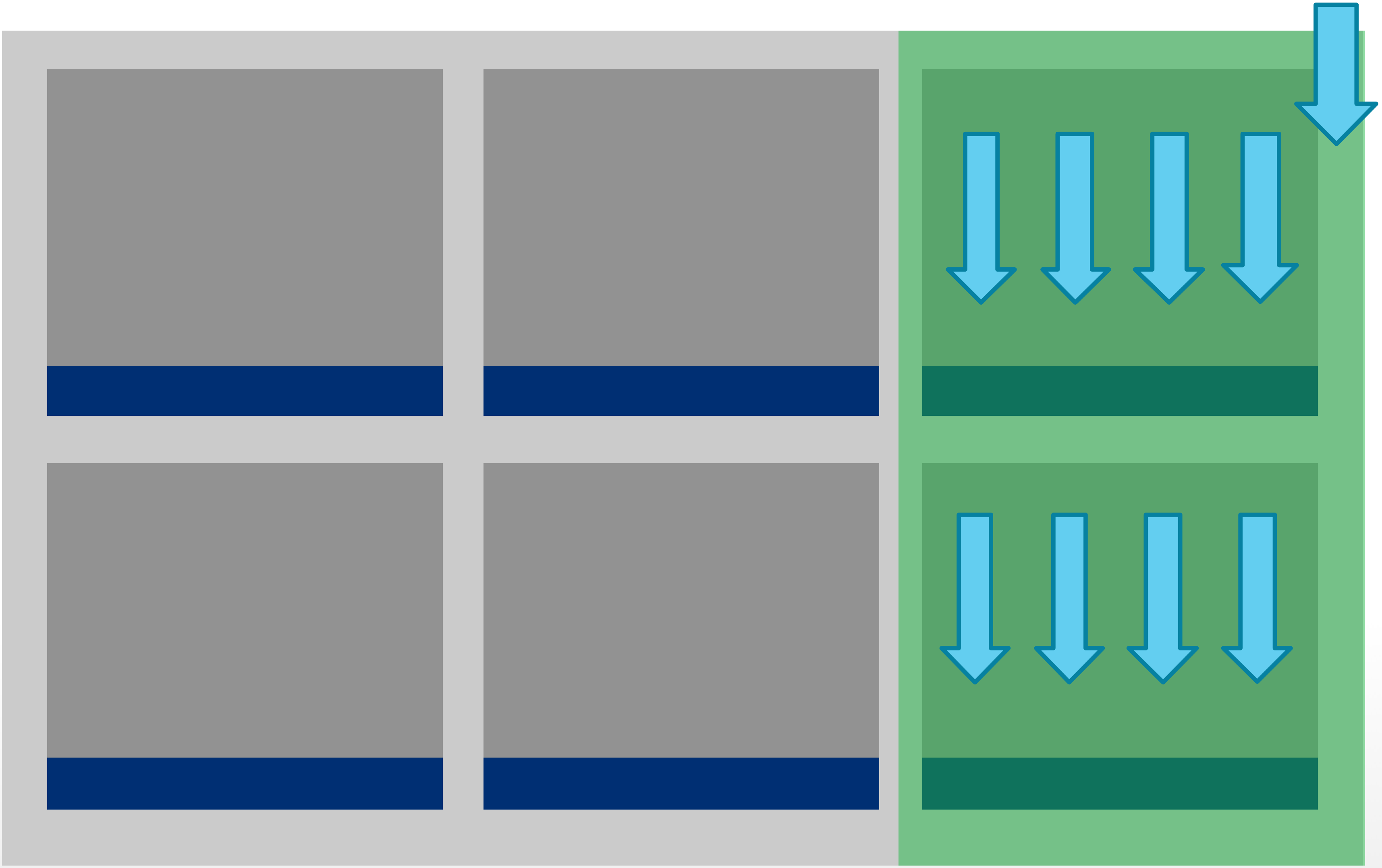
SMALL? PREDICTABLE?



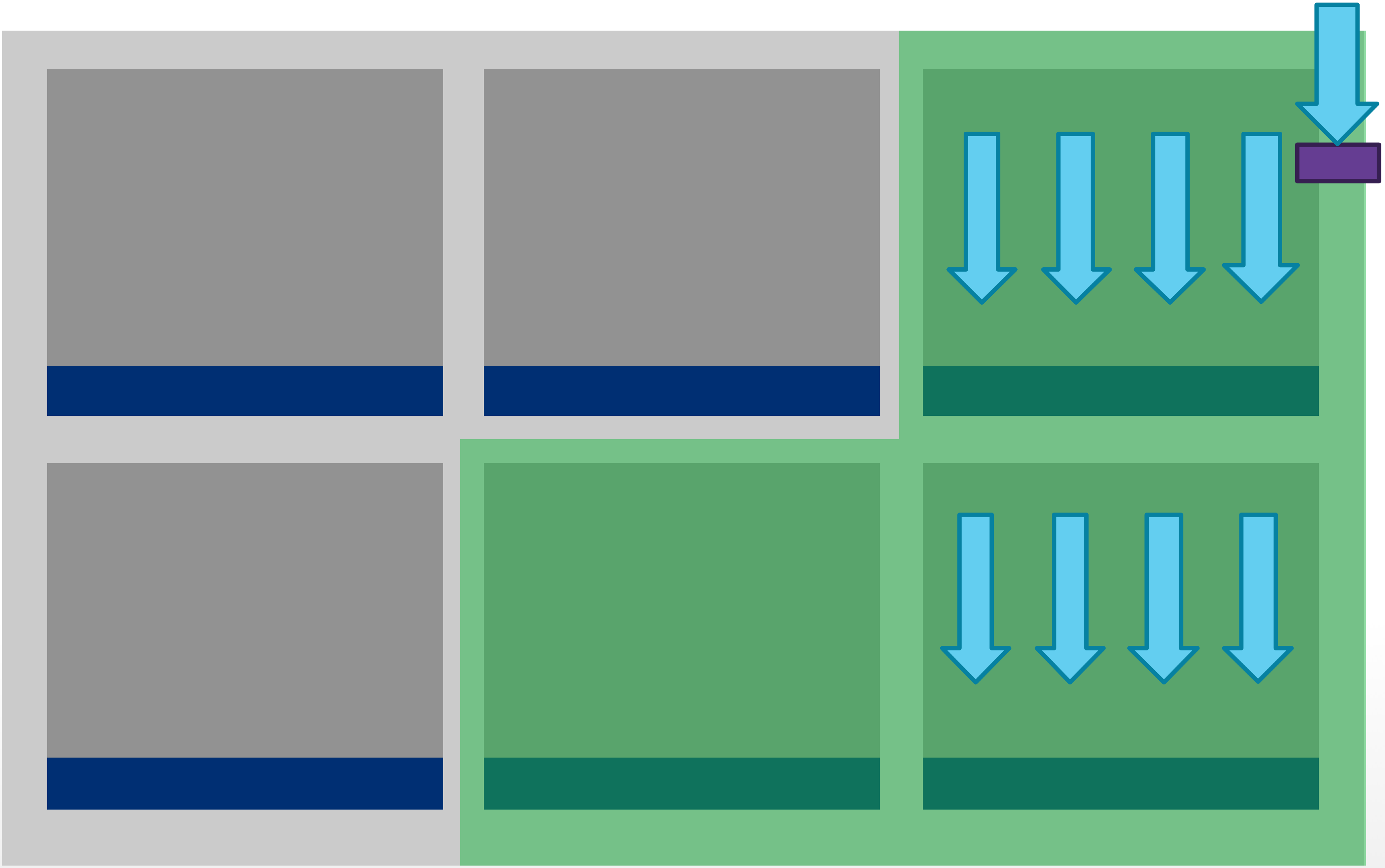
MOSIX: LOAD BALANCING



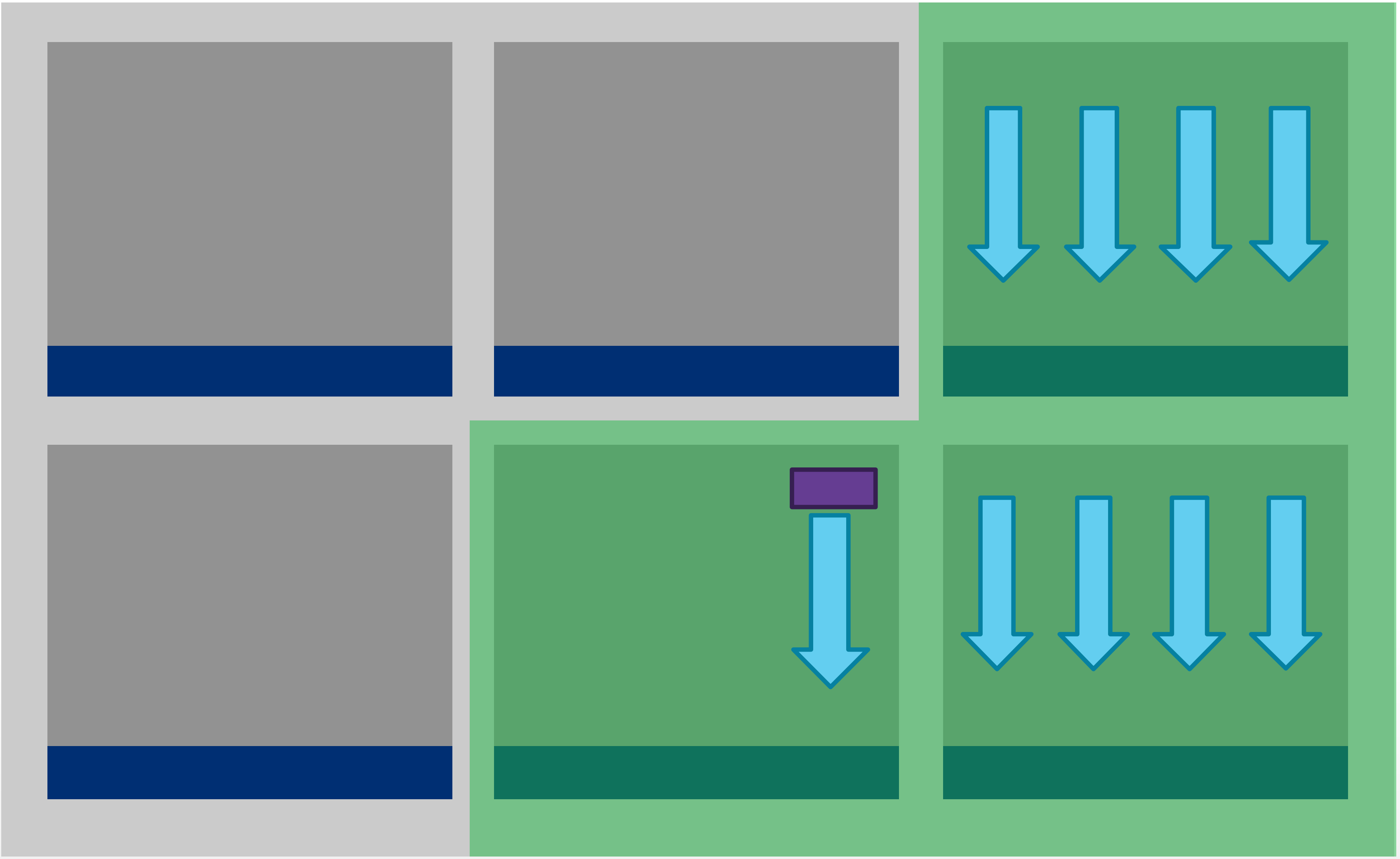
MOSIX: LOAD BALANCING



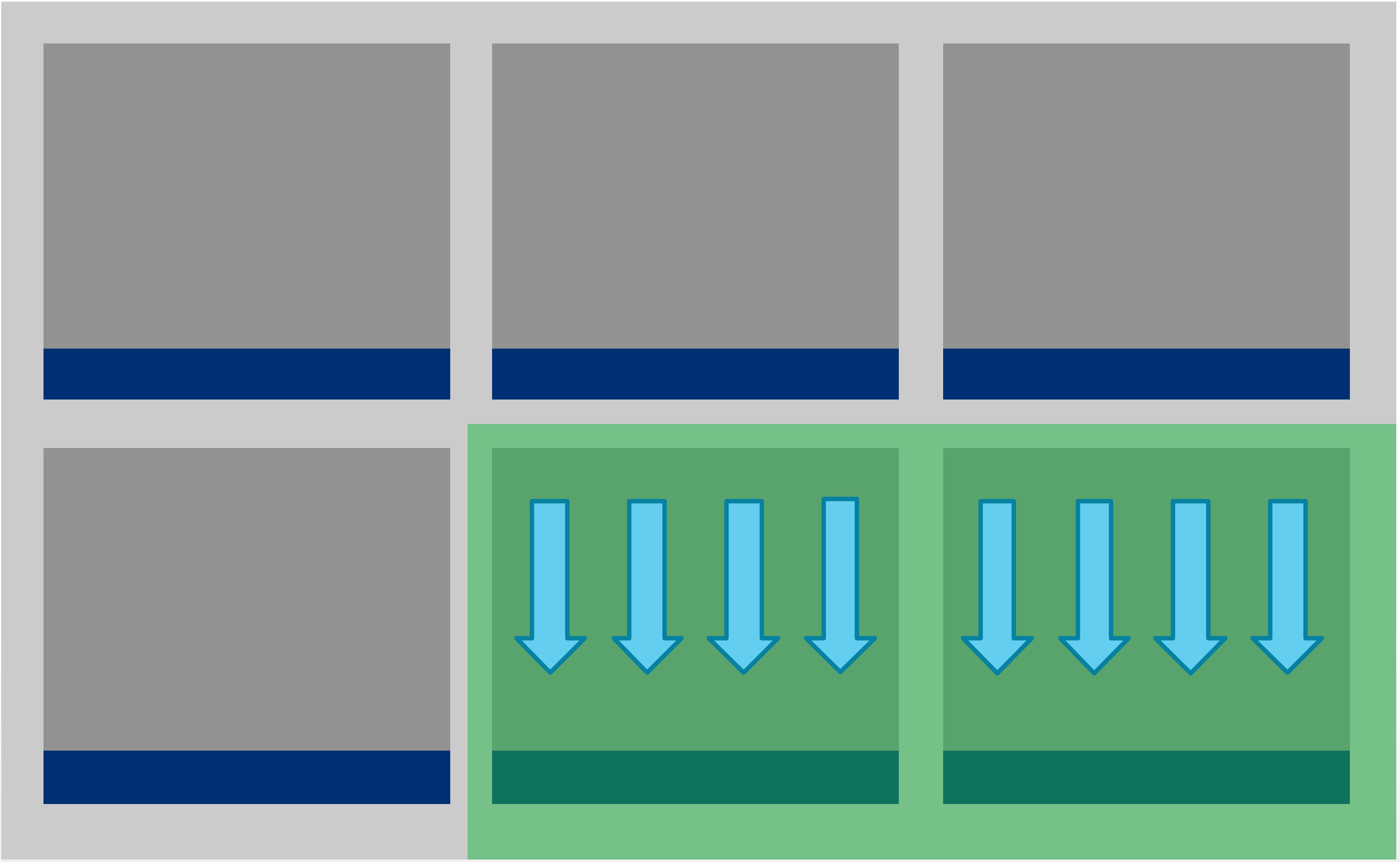
MOSIX: LOAD BALANCING



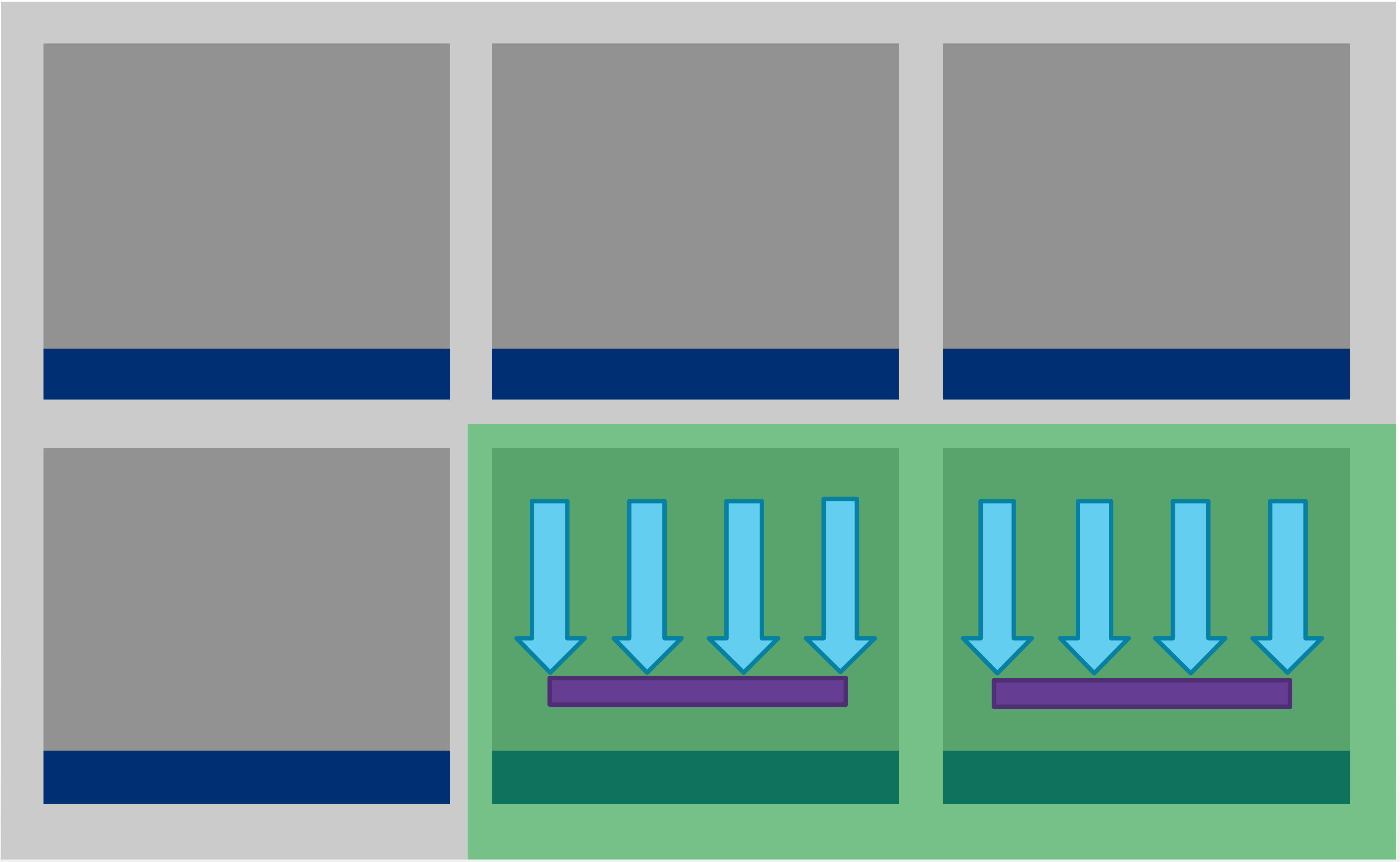
MOSIX: LOAD BALANCING



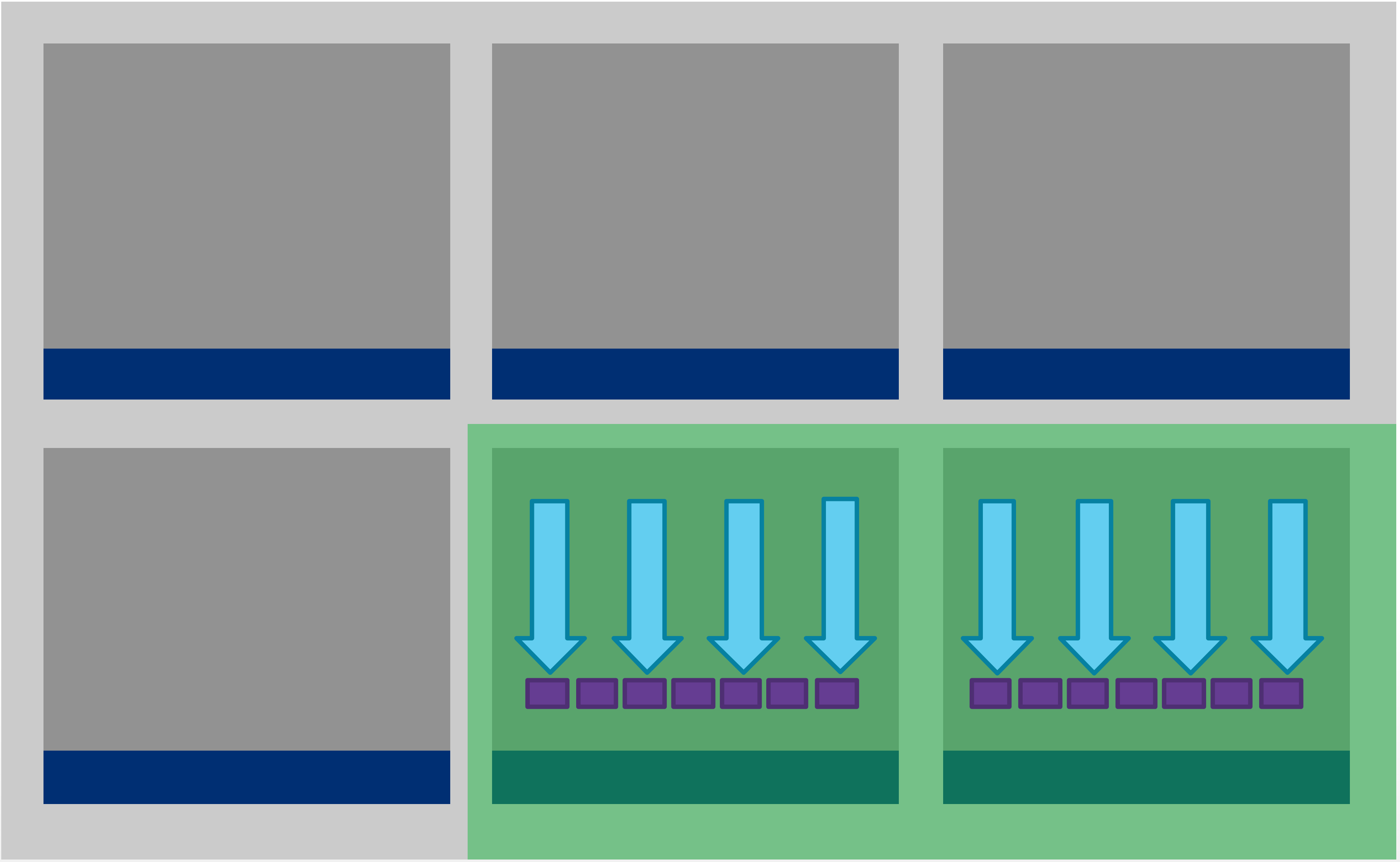
REDUNDANT CHECKPOINT



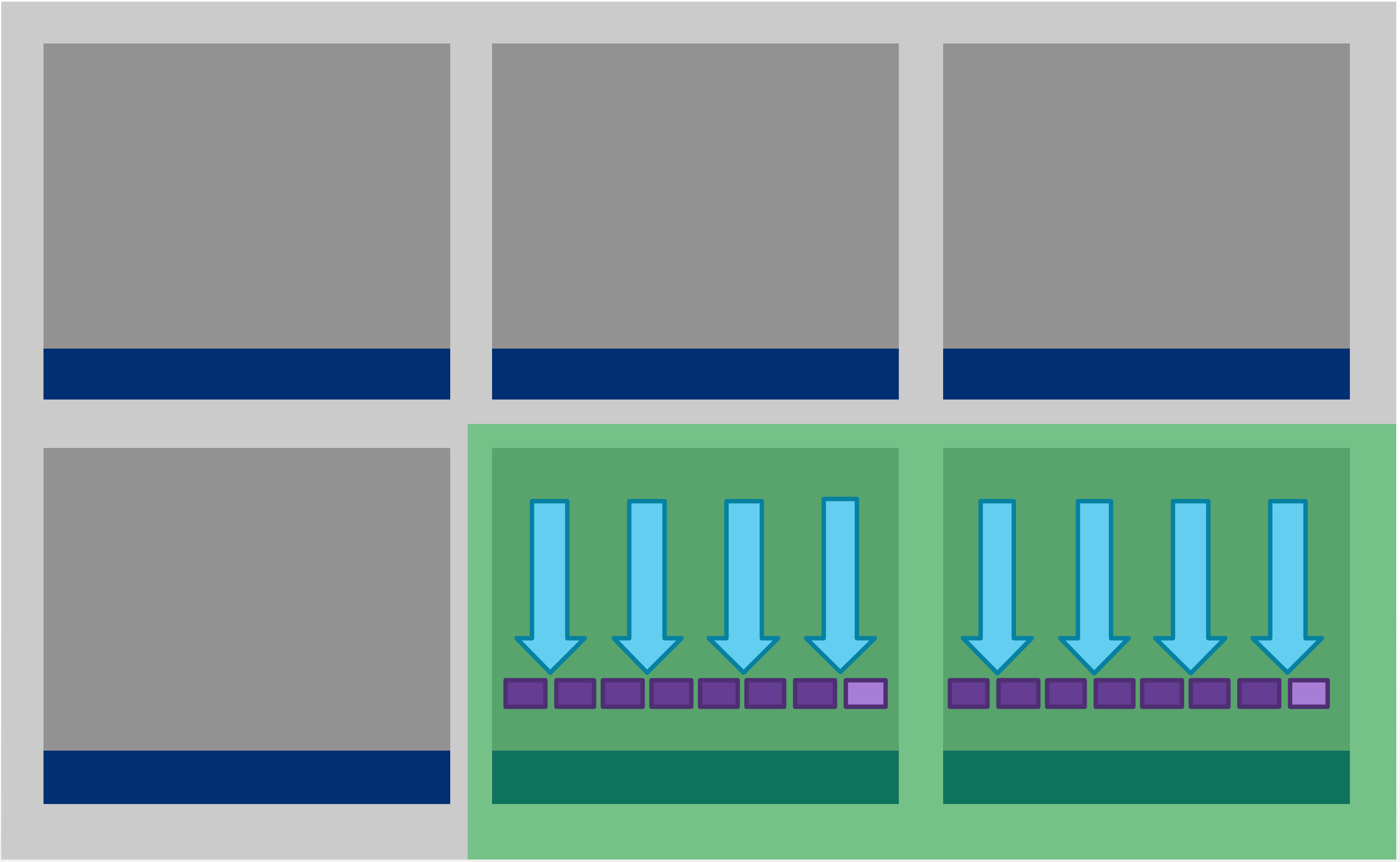
REDUNDANT CHECKPOINT



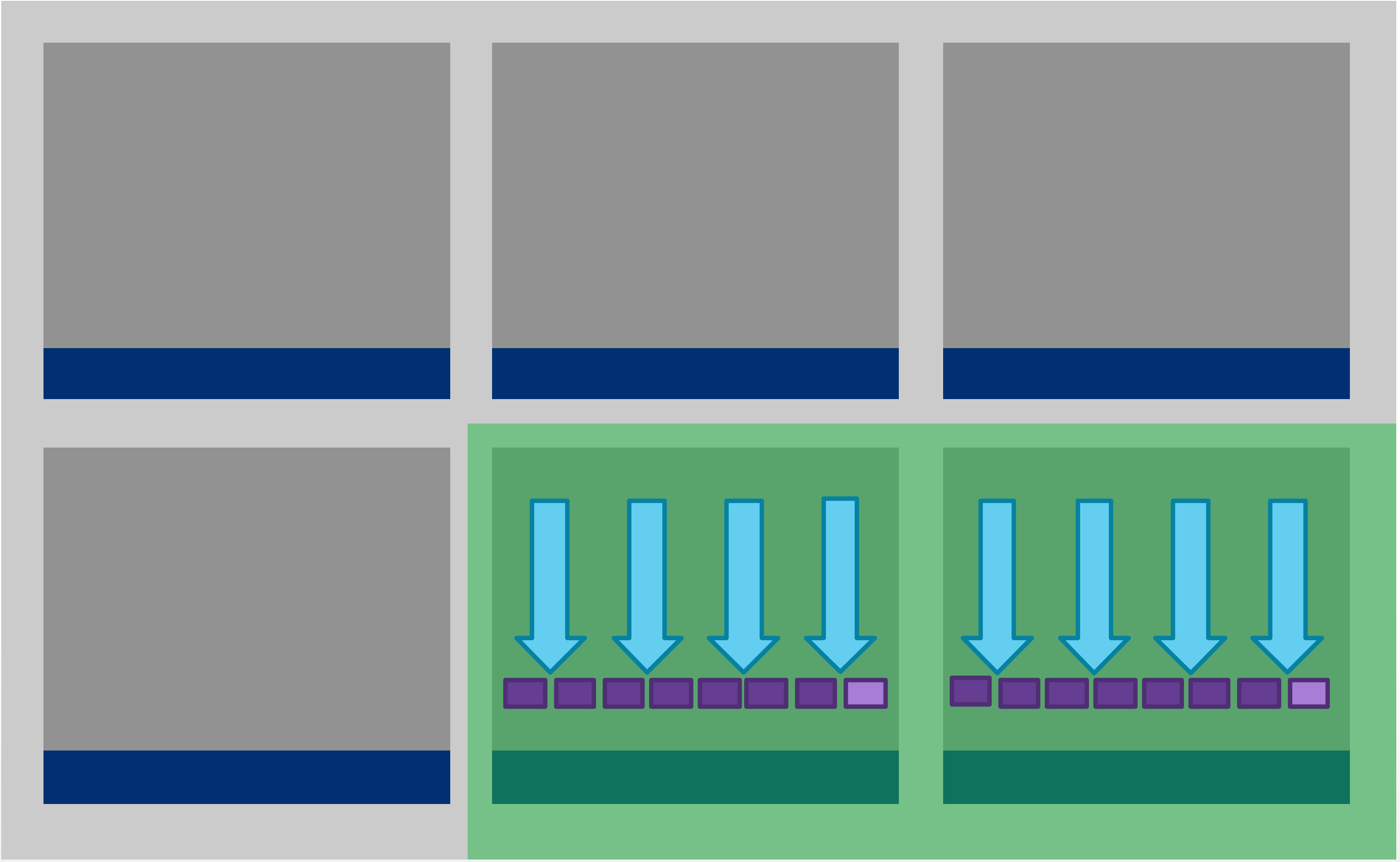
REDUNDANT CHECKPOINT



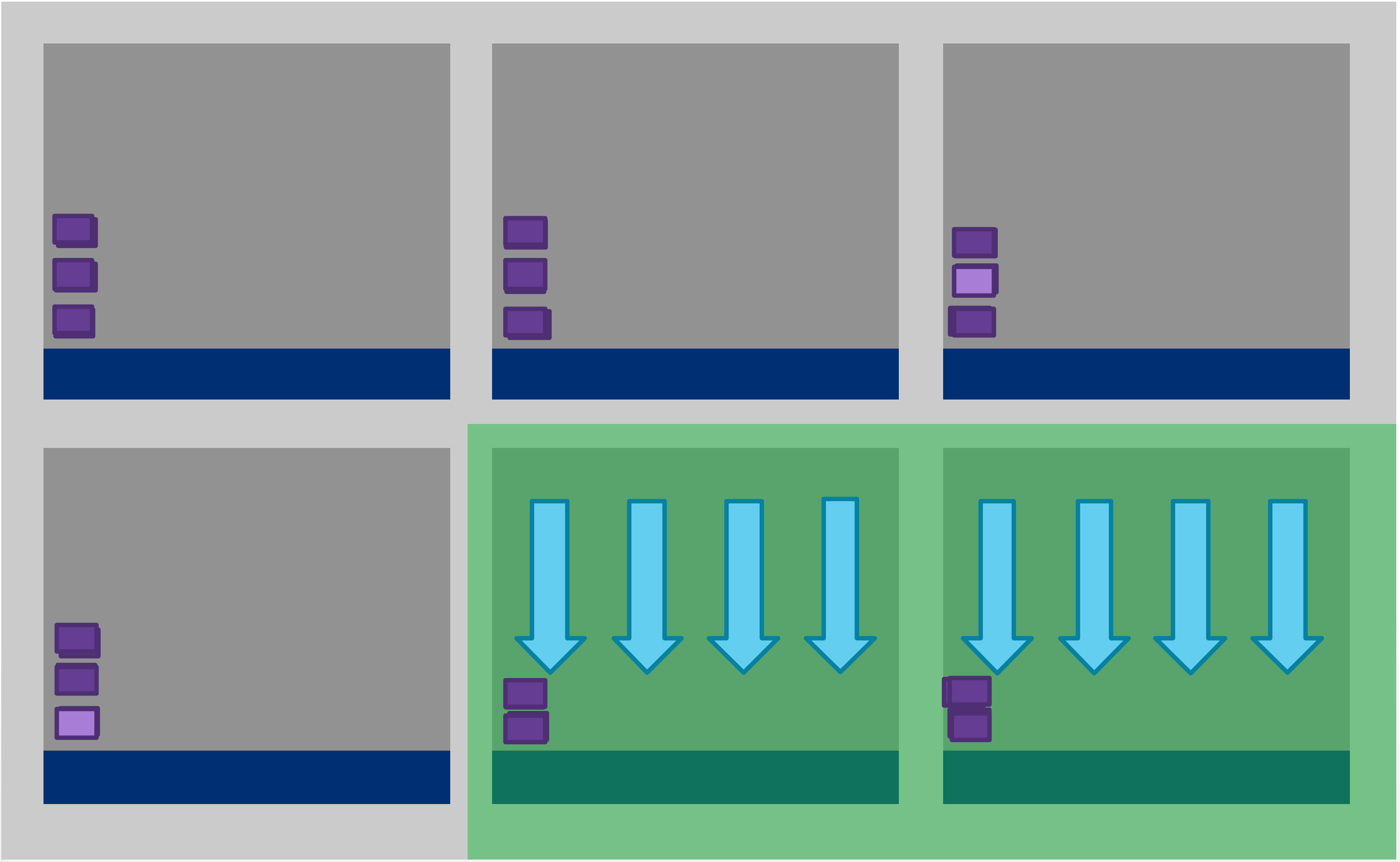
REDUNDANT CHECKPOINT



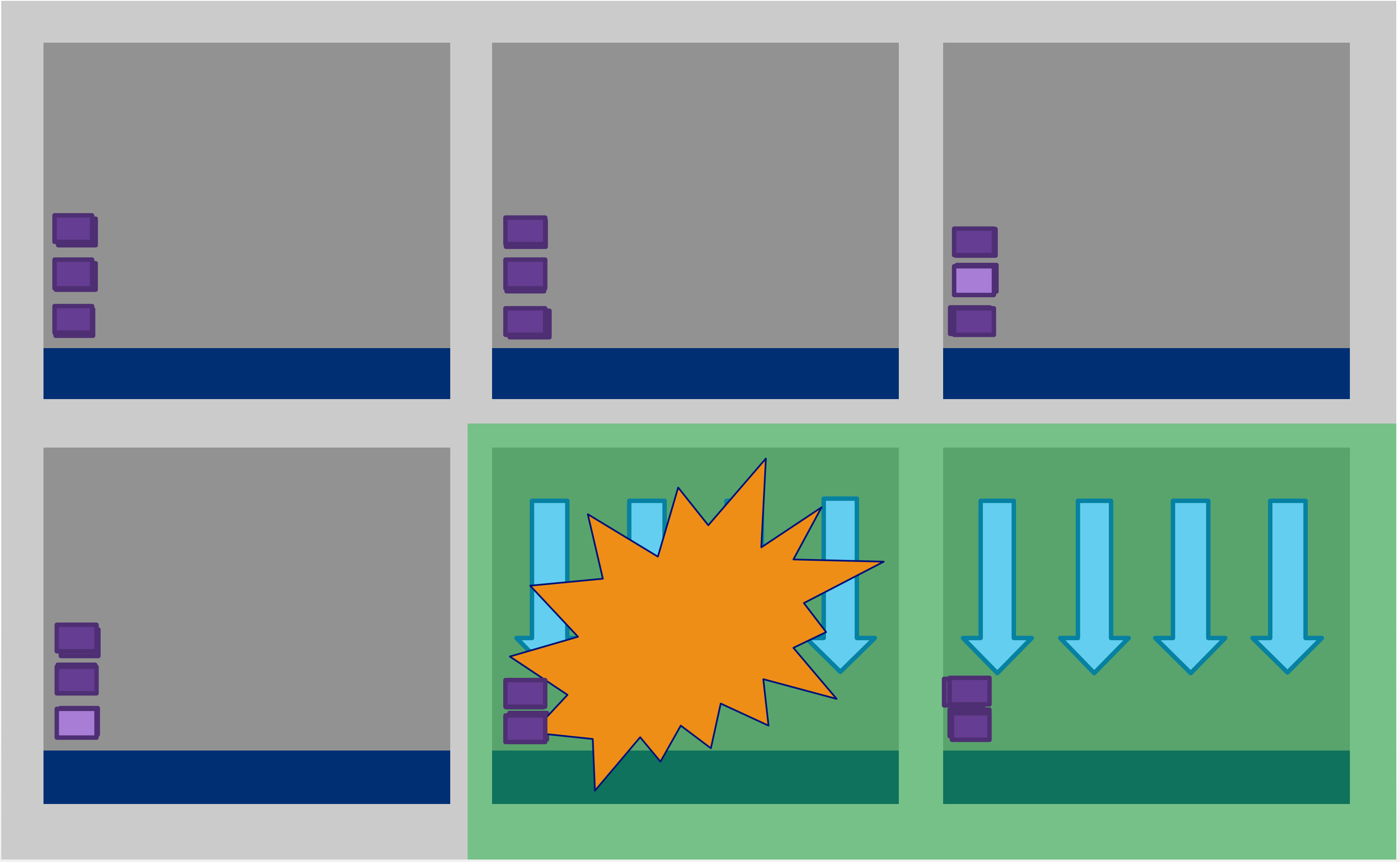
REDUNDANT CHECKPOINT



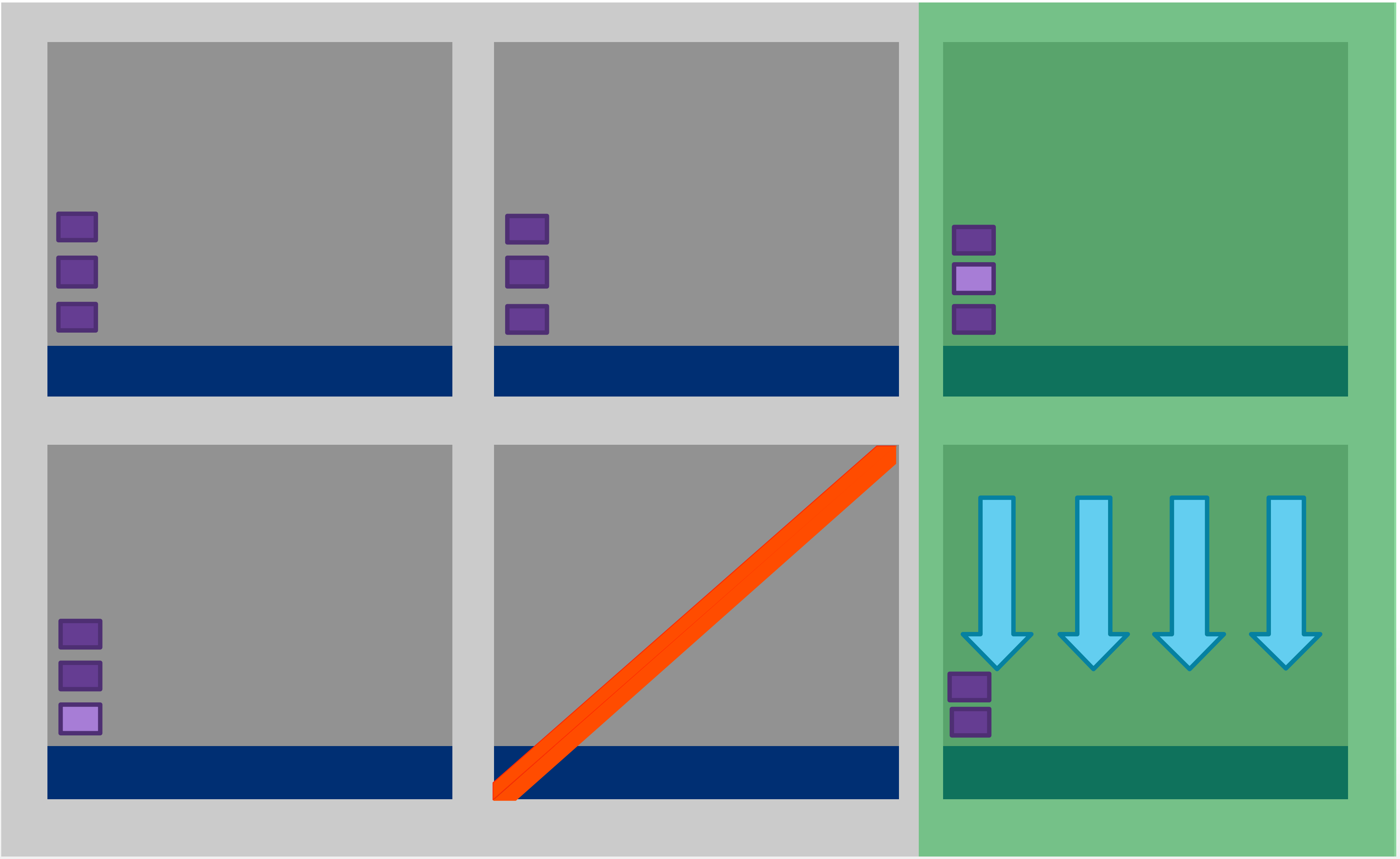
REDUNDANT CHECKPOINT



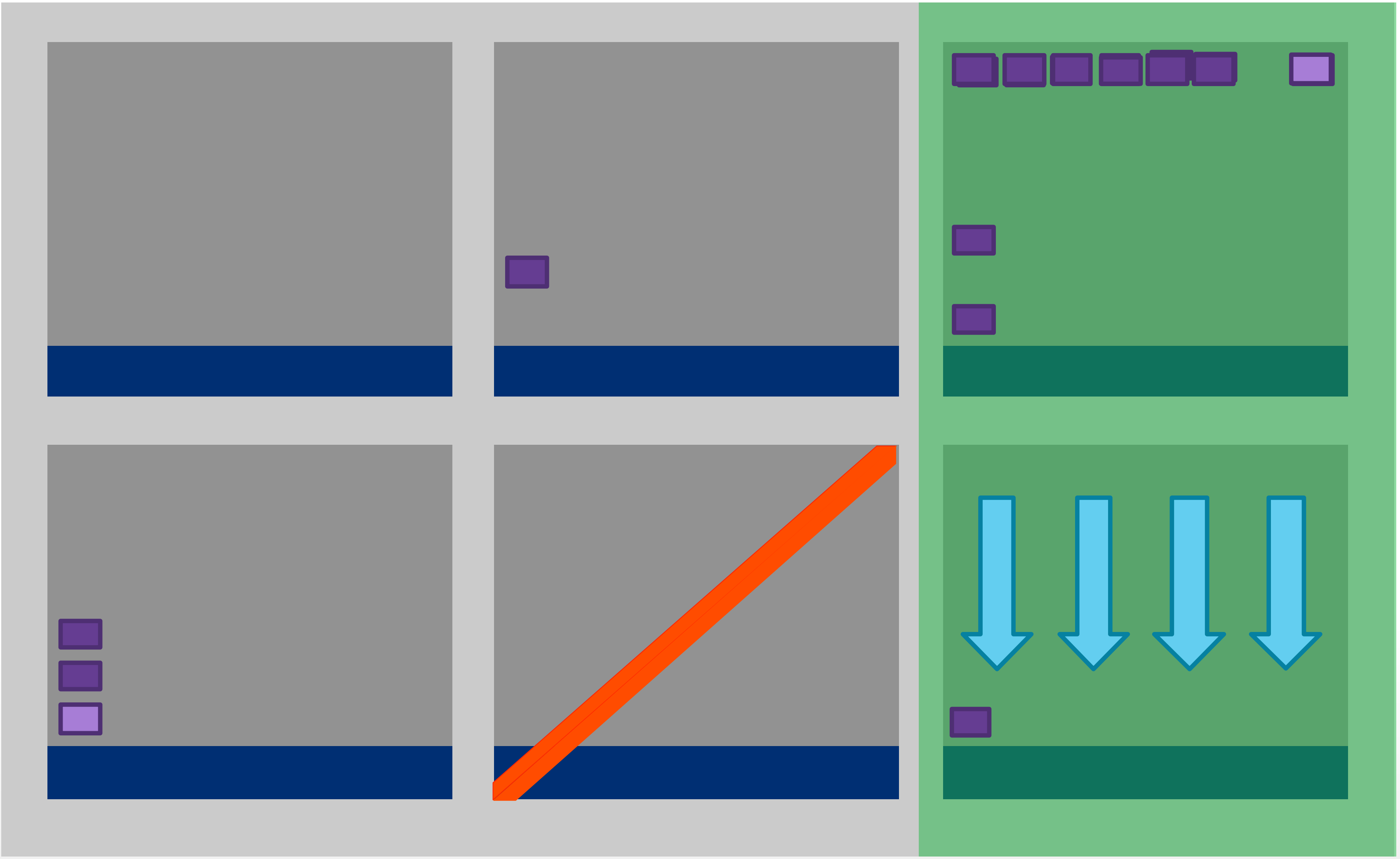
REDUNDANT CHECKPOINT



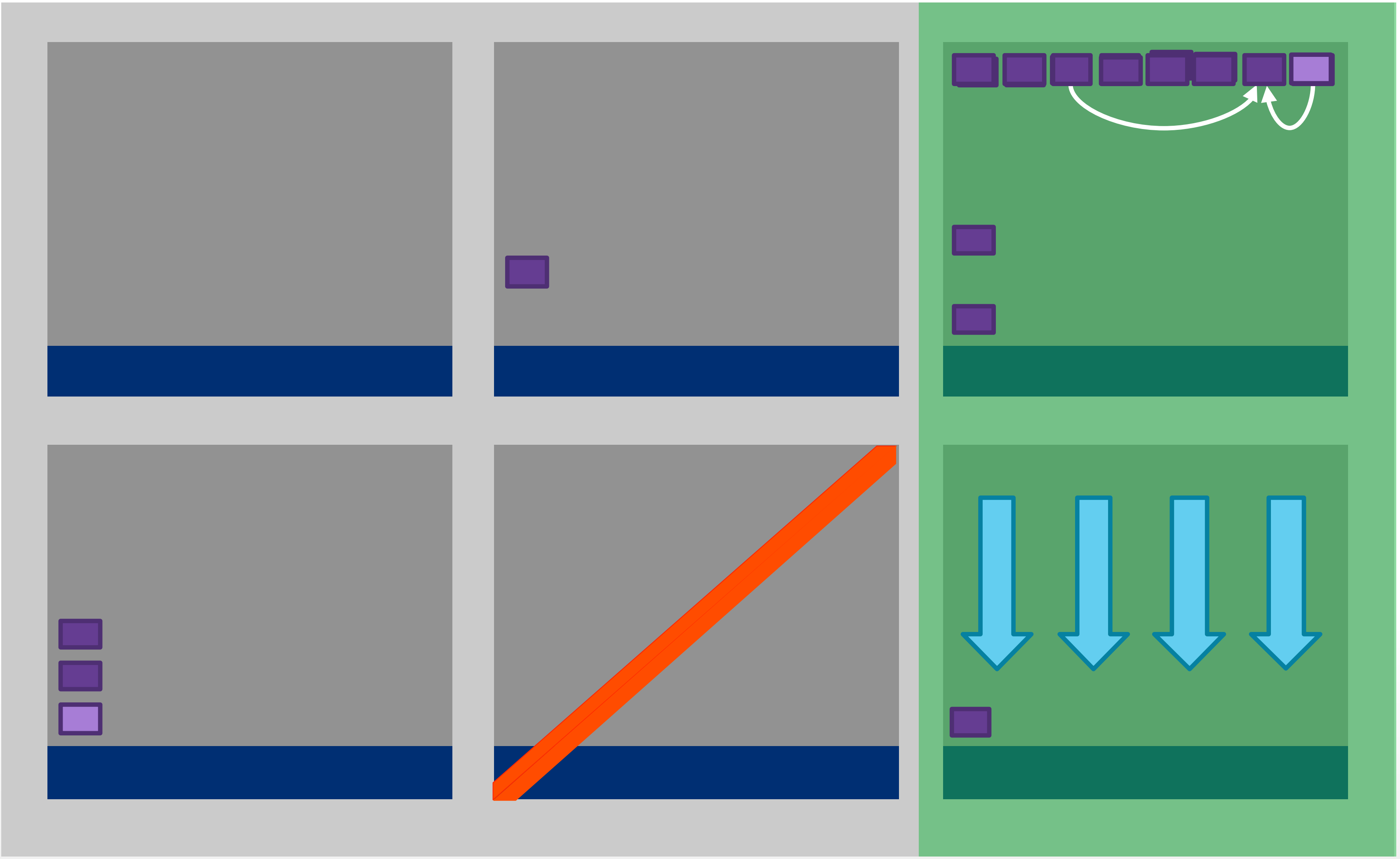
REDUNDANT CHECKPOINT



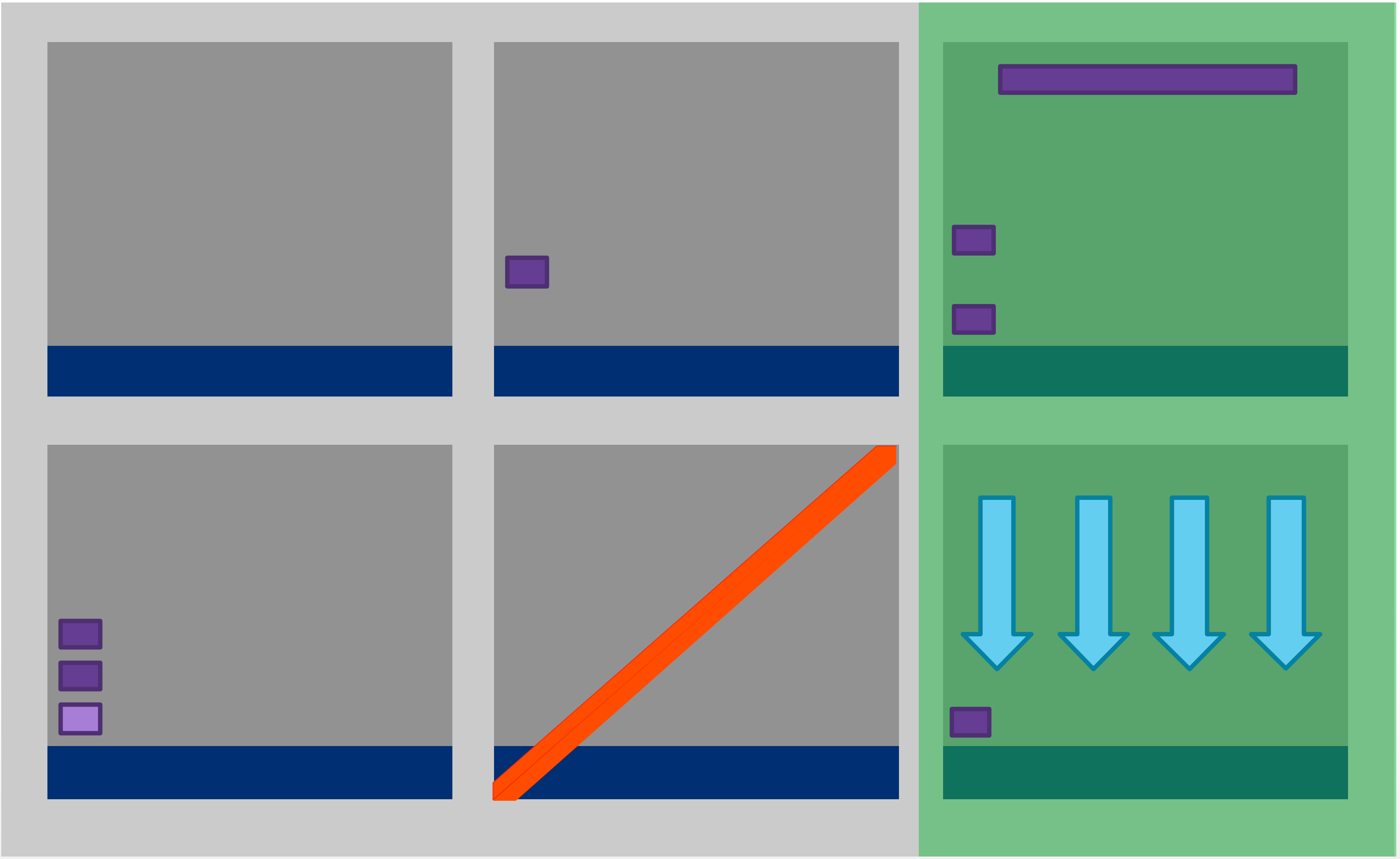
REDUNDANT CHECKPOINT



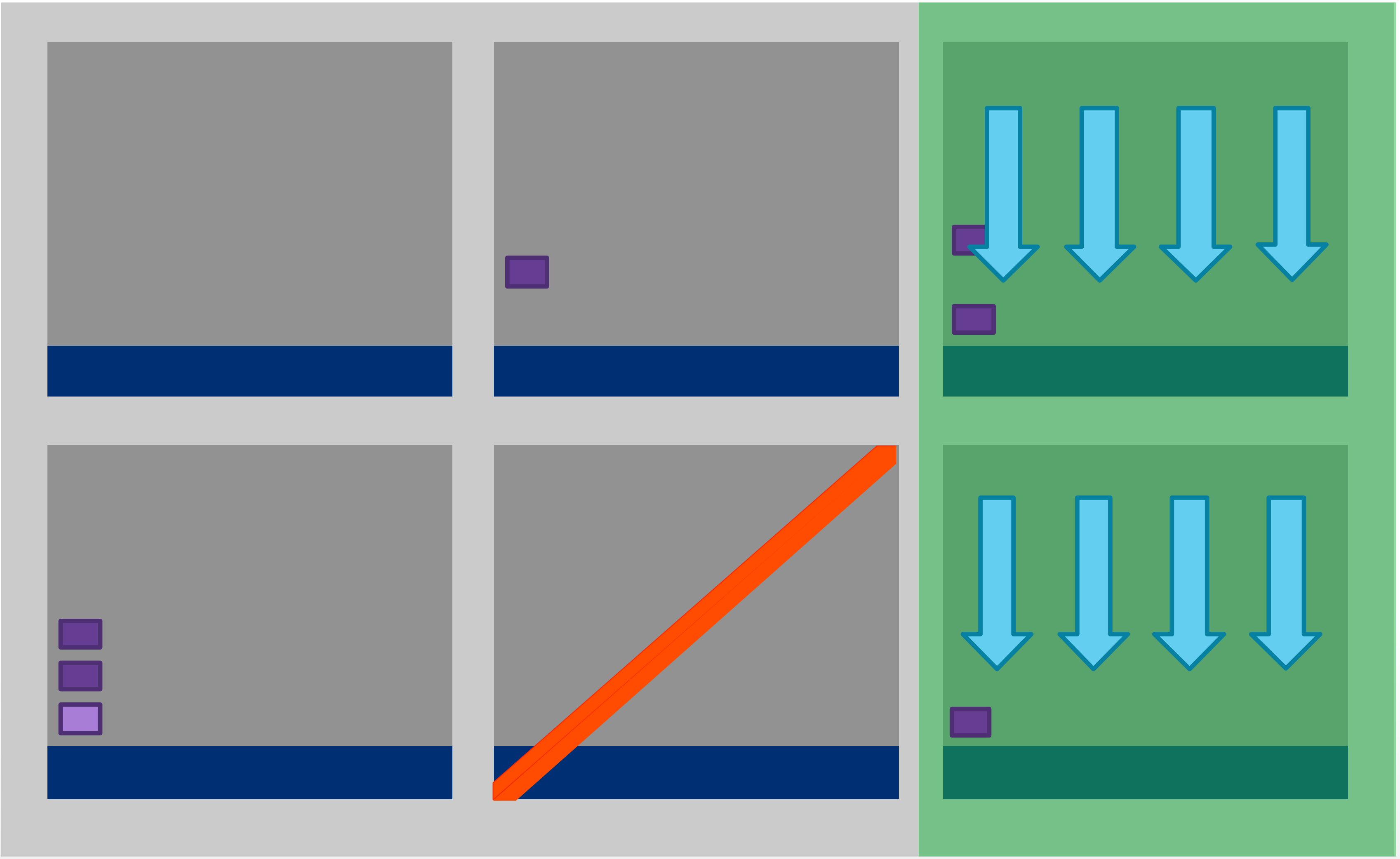
REDUNDANT CHECKPOINT



REDUNDANT CHECKPOINT

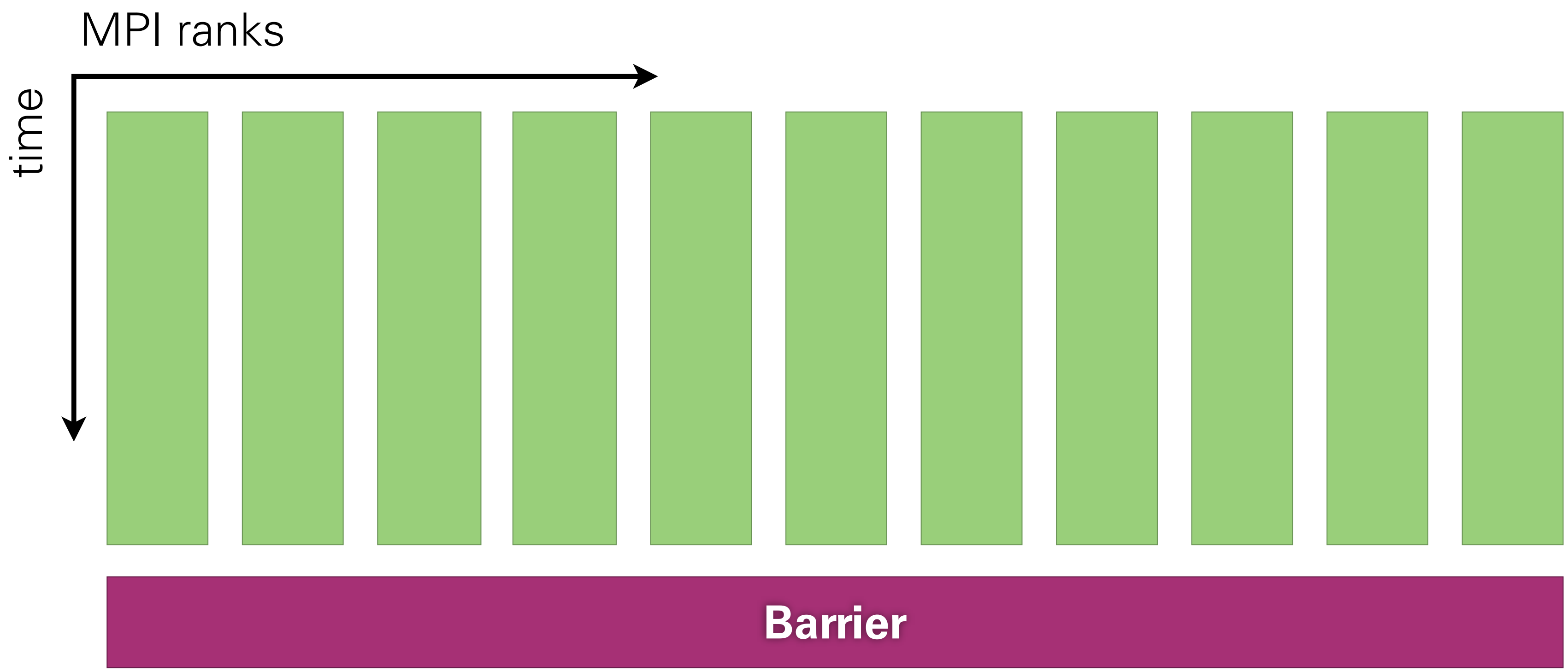


REDUNDANT CHECKPOINT

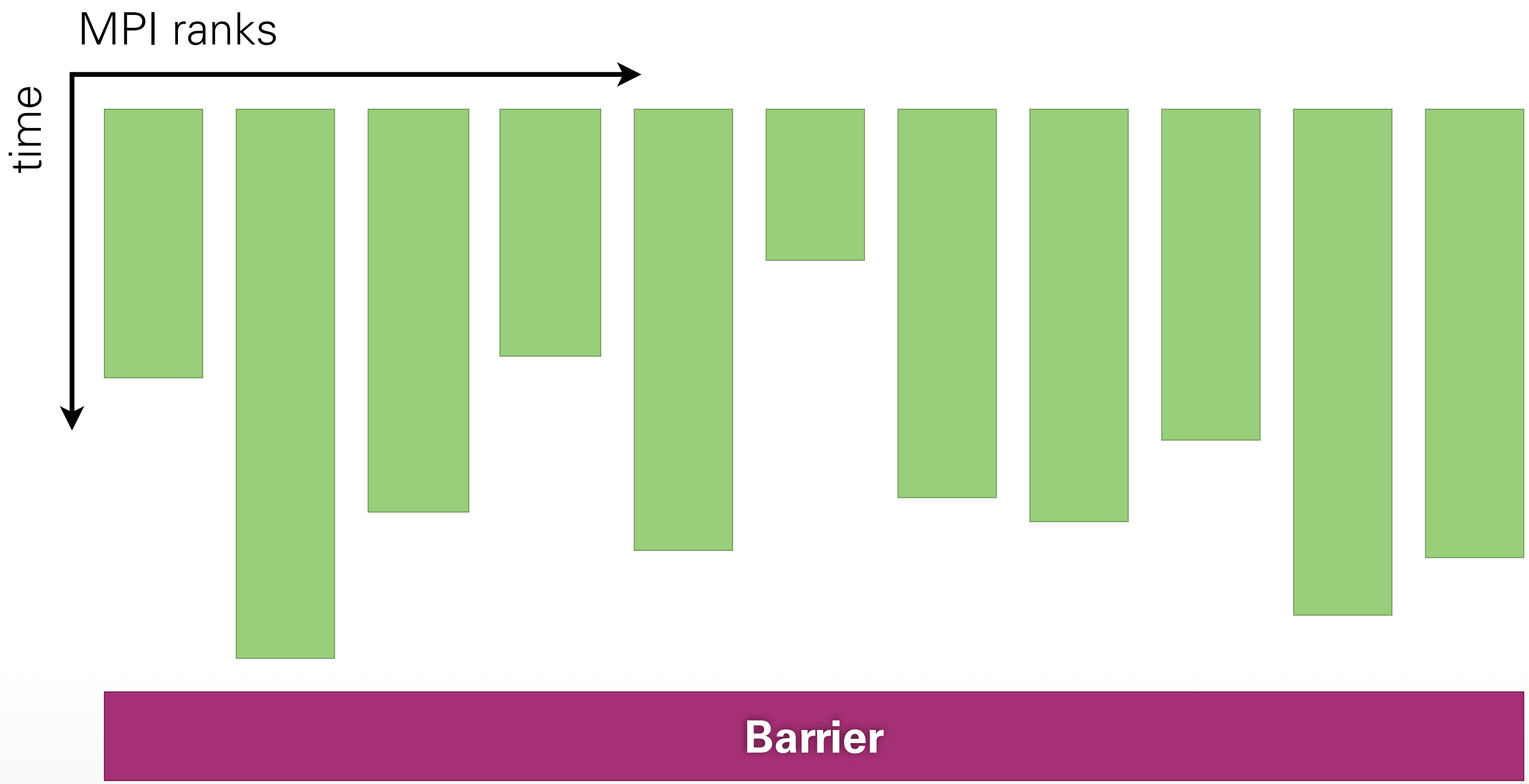


EXPERIMENTS: IMBALANCES, OVERDECOMPOSITION AND OVERSUBSCRIPTION

TOWARDS BALANCING

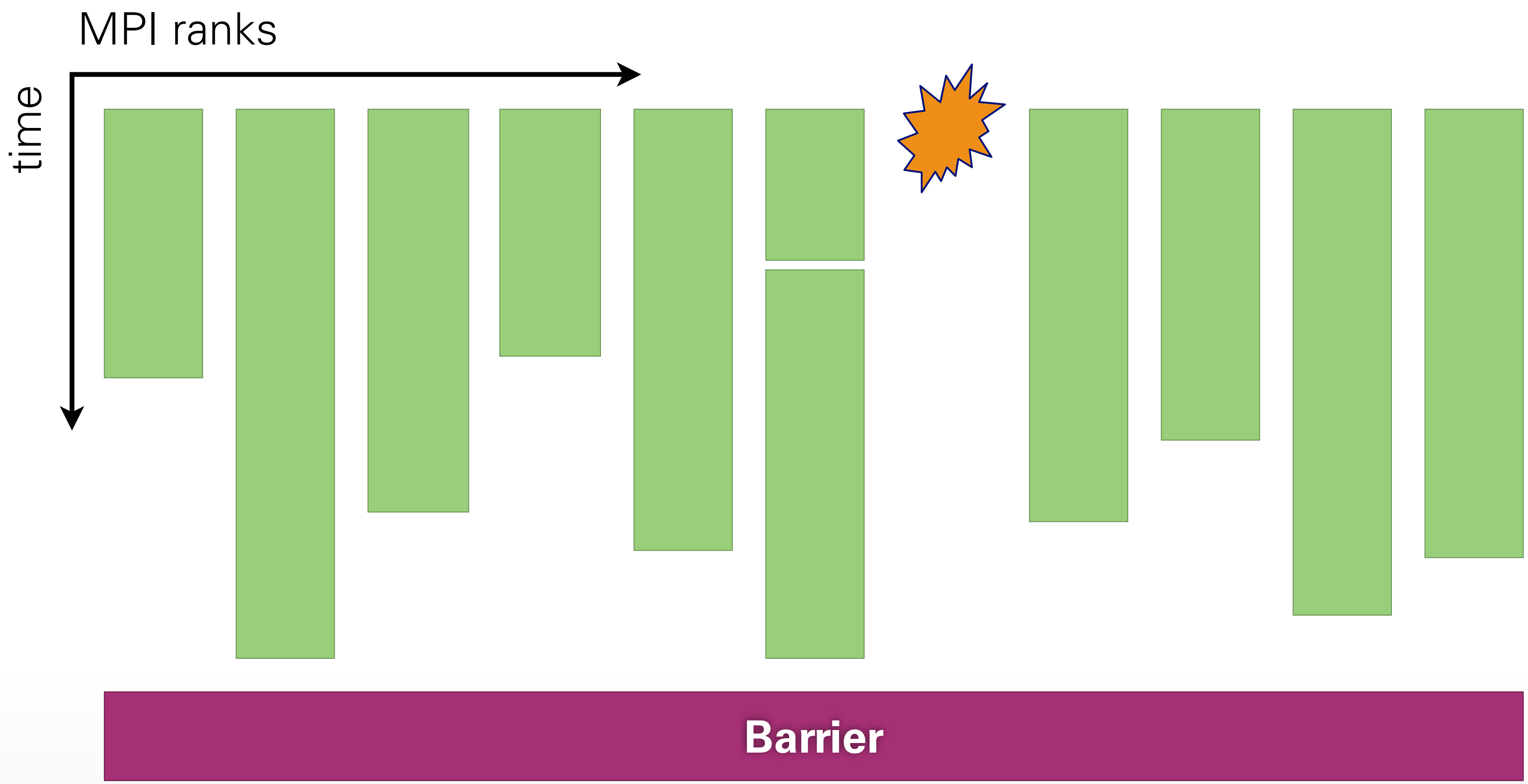


“MESSY” HPC



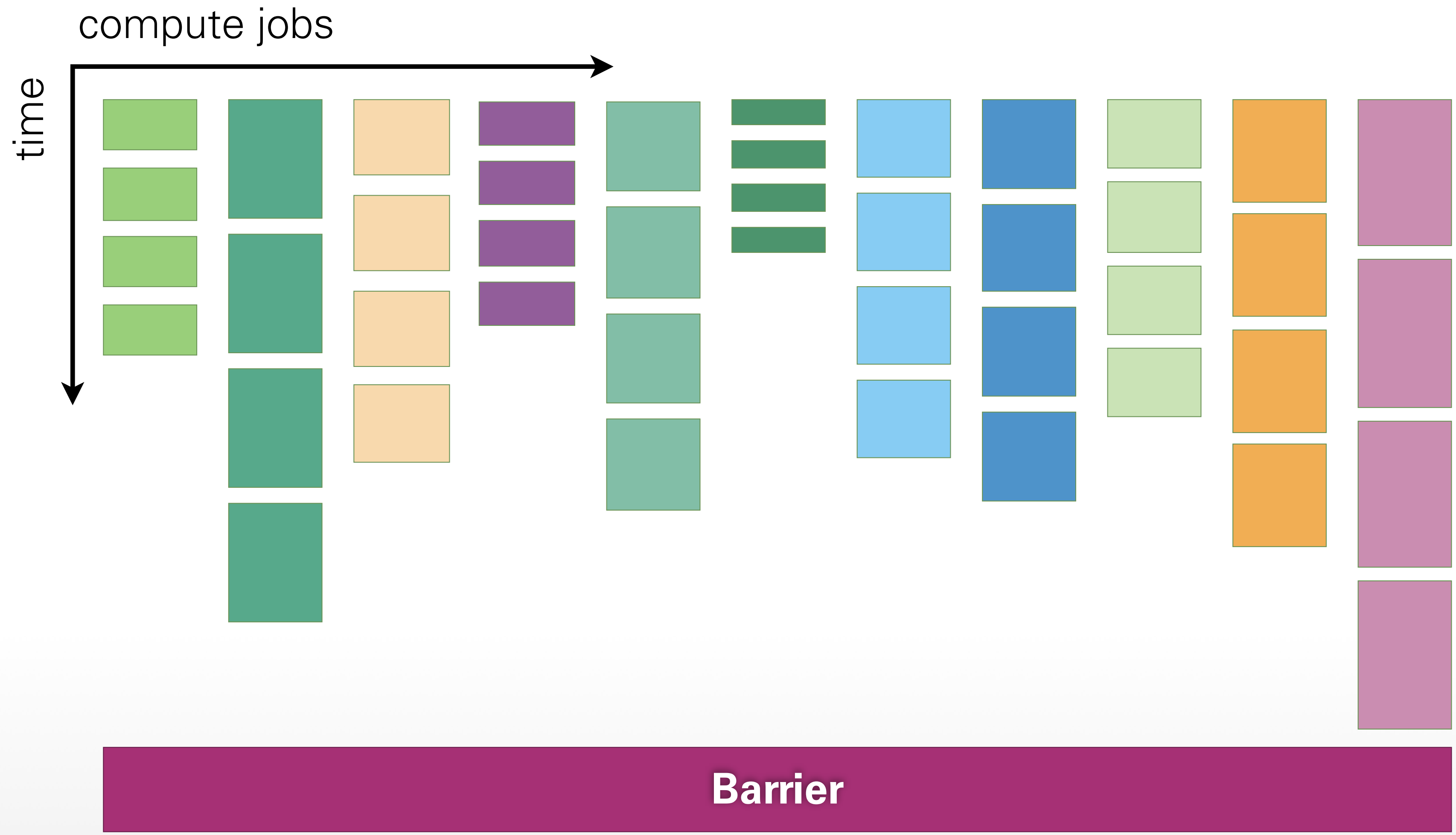
Imbalance in application workload

FAILURES



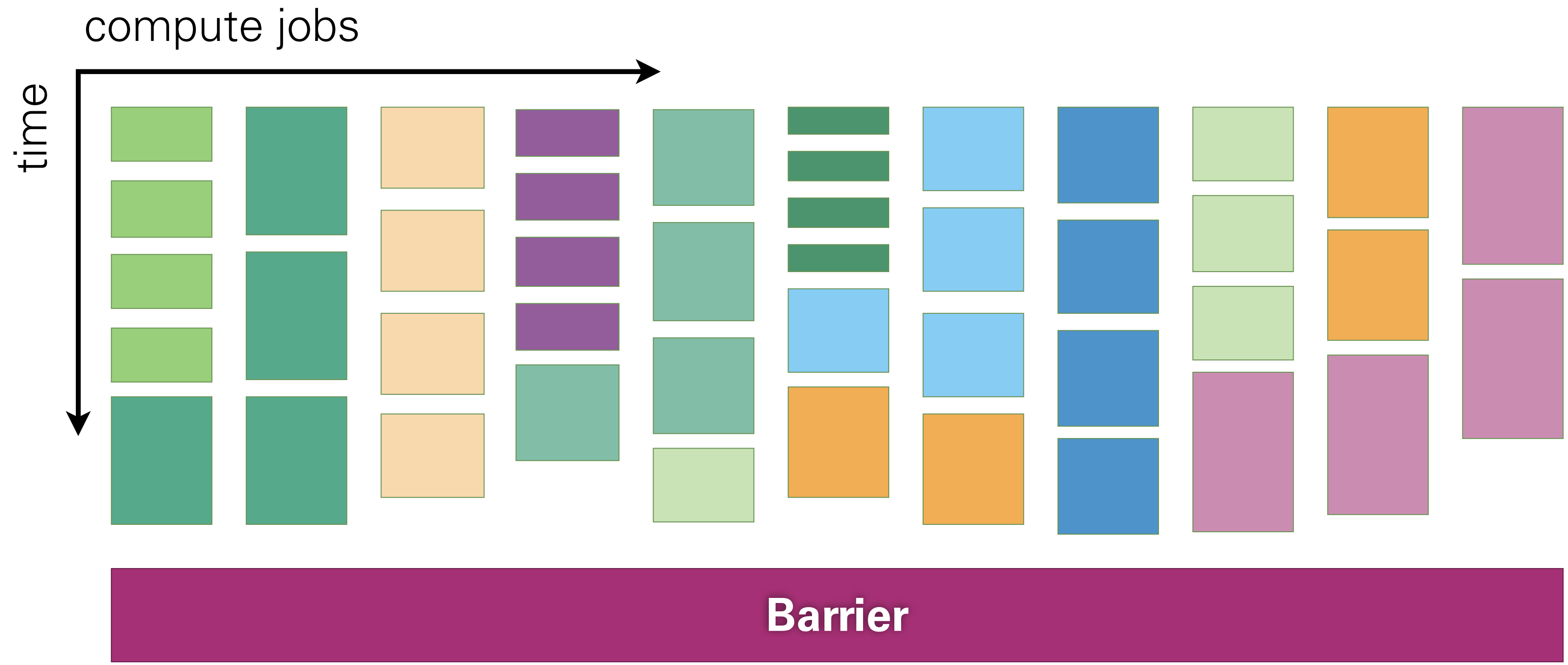
Reassign work to react to node failure

SPLITTING BIG JOBS



overdecomposition & “oversubscription”

SMALL JOBS (NO DEPS)



Execute small jobs in parallel (if possible)

IMBALANCES

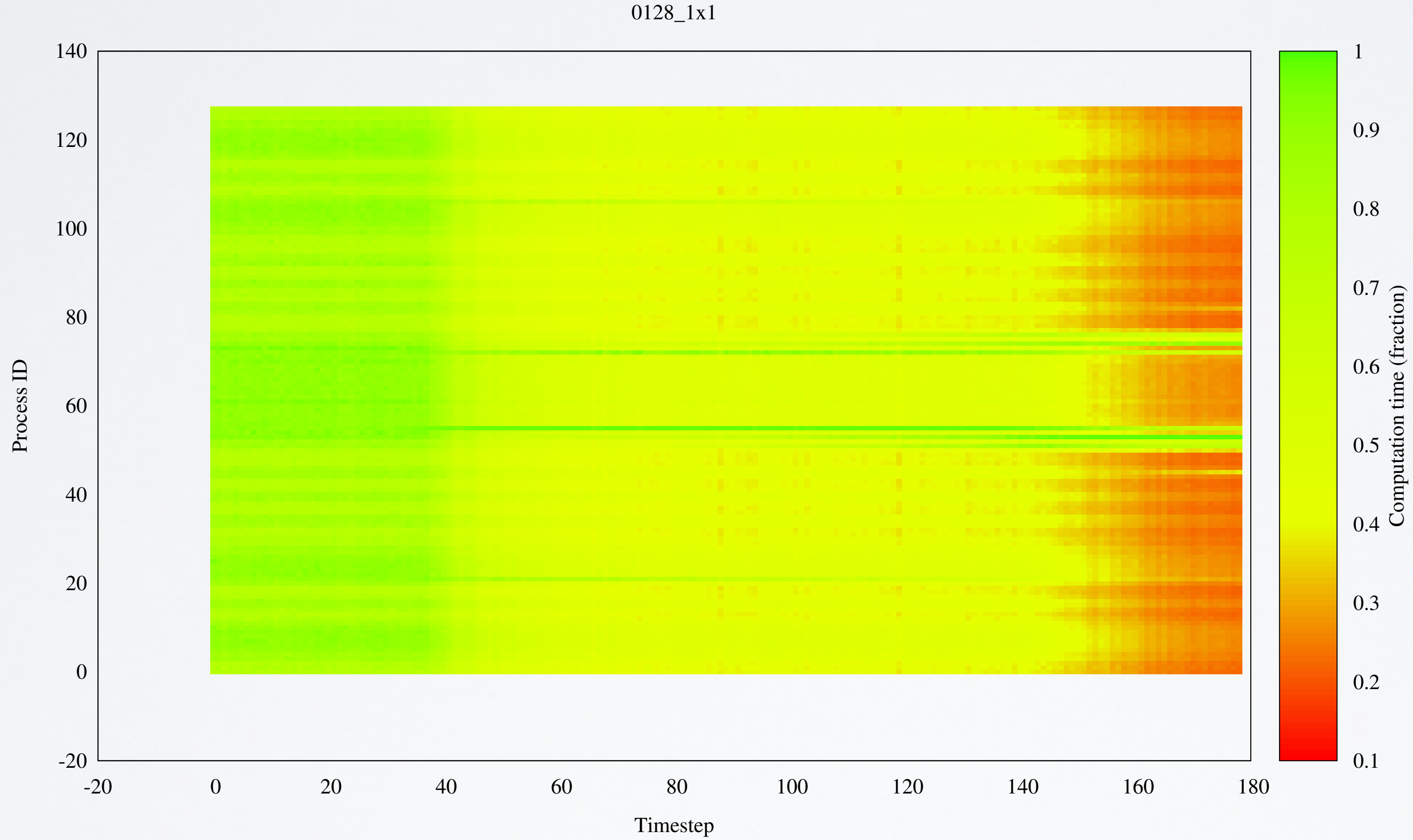
Application: COSMO-SPECS+FD4

Unbalanced compute times of ranks per time step

Application: COSMO-SPECS+FD4

Balanced compute times of ranks per time step

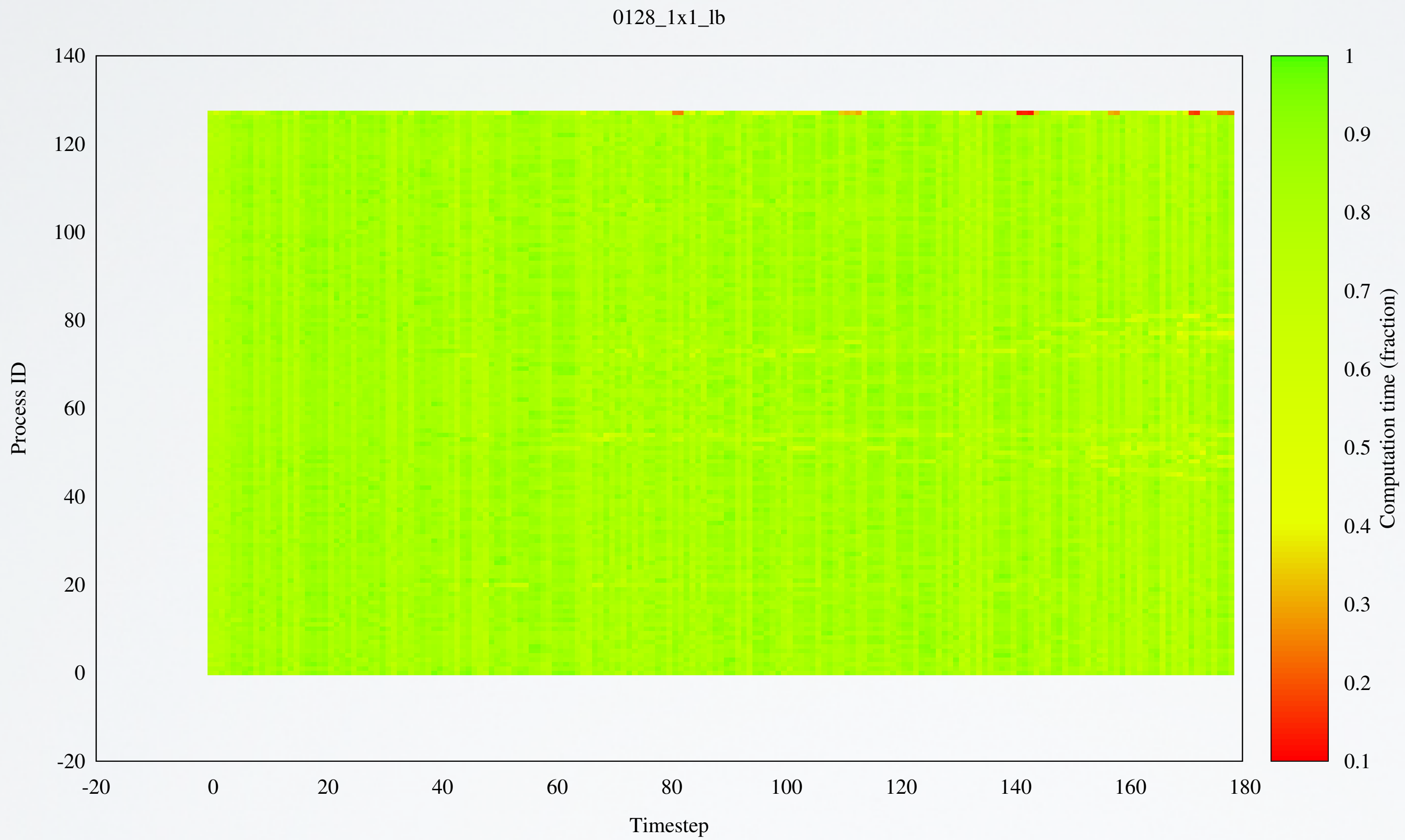
IMBALANCES



Unbalanced compute times of ranks per time step

Application: COSMO-SPECS+FD4

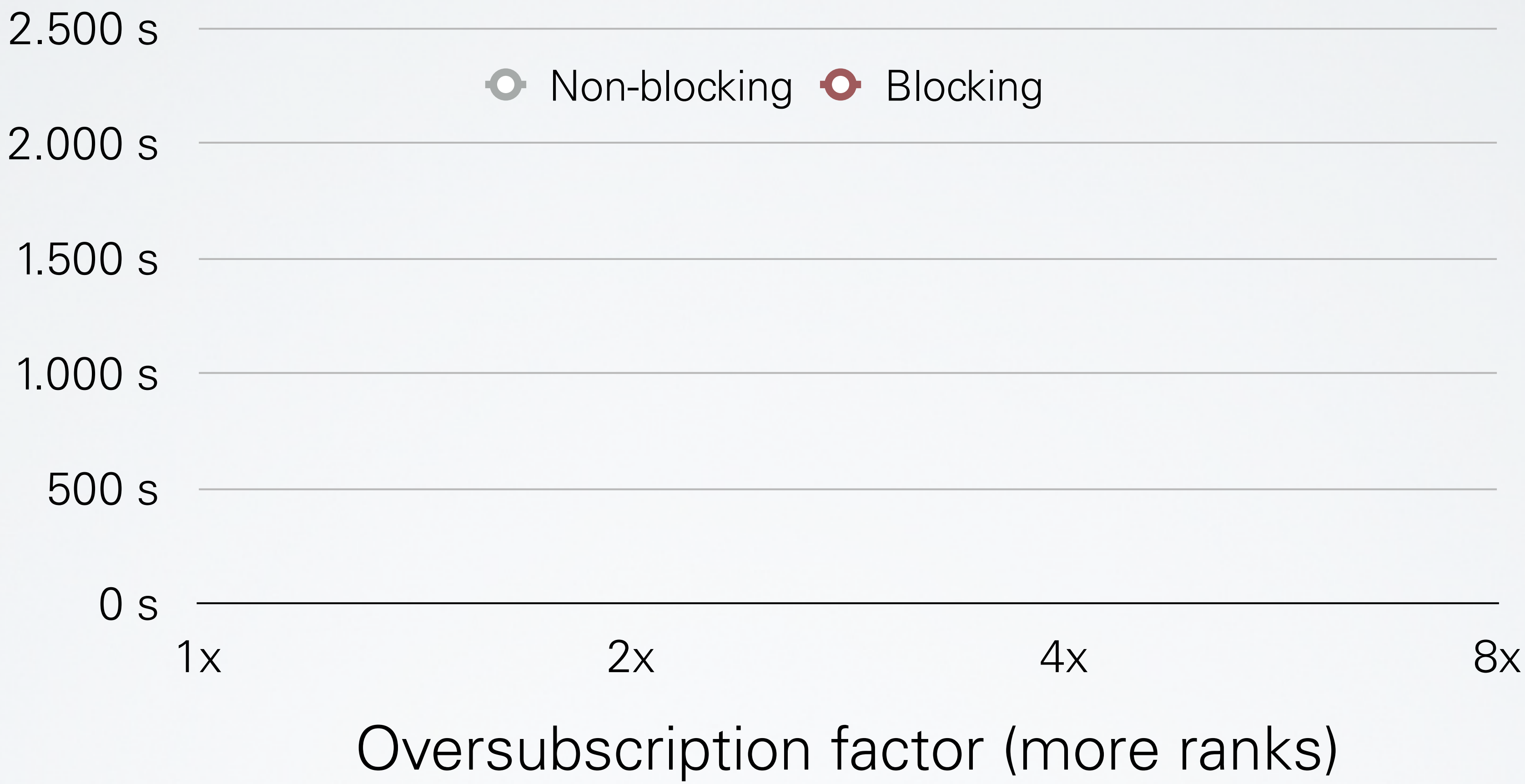
IMBALANCES



Balanced compute times of ranks per time step

Application: COSMO-SPECS+FD4

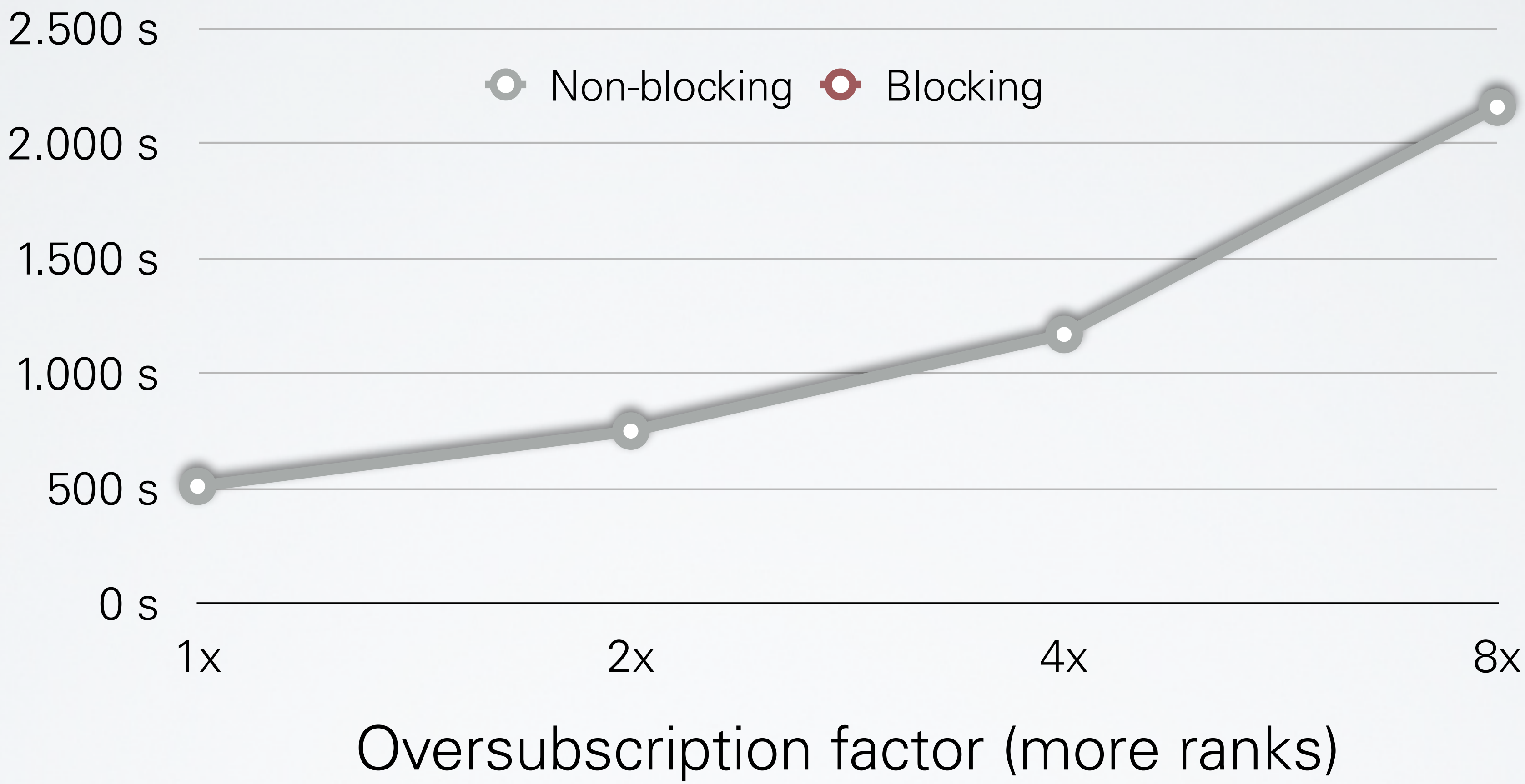
OVERSUBSCRIPTION



Application: COSMO-SPECS+FD4 (no load balancing)

- Taurus 16 nodes w/ 16 Xeon E5-2690 (Sandy Bridge) @ 2.90GHz
- 1x - 8x oversubscription (256 - 2048 MPI ranks, same problem size)

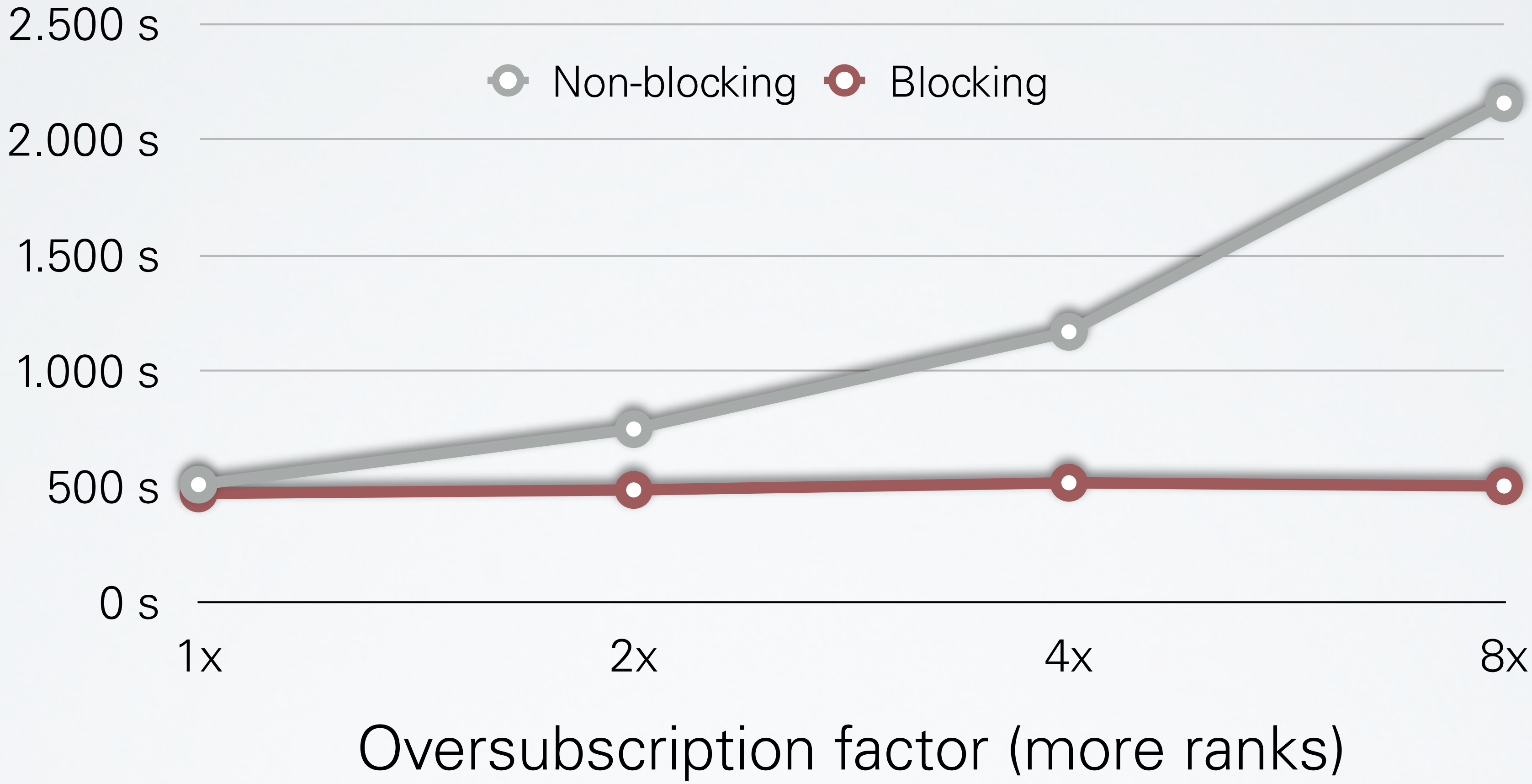
OVERSUBSCRIPTION



Application: COSMO-SPECS+FD4 (no load balancing)

- Taurus 16 nodes w/ 16 Xeon E5-2690 (Sandy Bridge) @ 2.90GHz
- 1x - 8x oversubscription (256 - 2048 MPI ranks, same problem size)

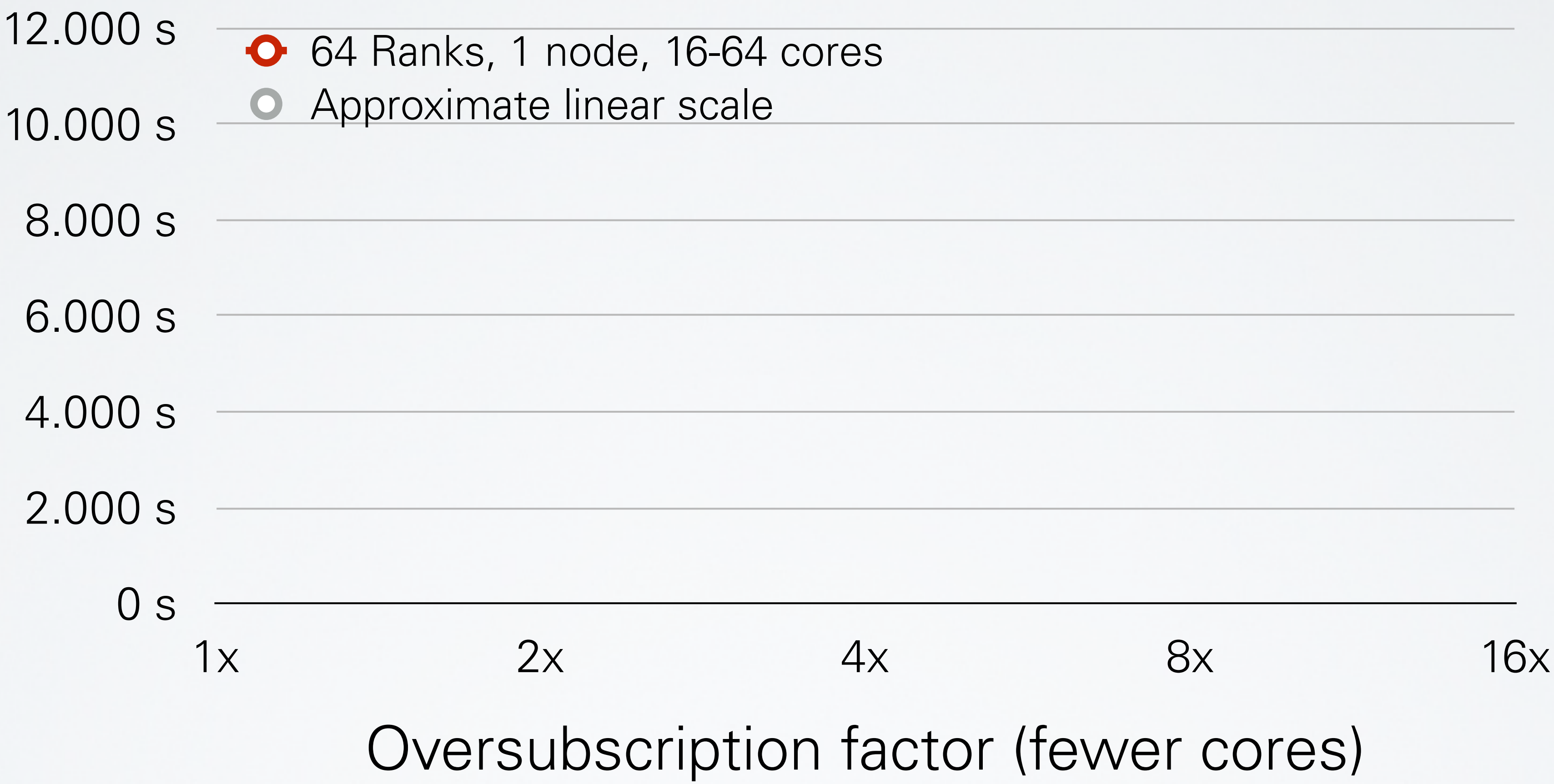
OVERSUBSCRIPTION



Application: COSMO-SPECS+FD4 (no load balancing)

- Taurus 16 nodes w/ 16 Xeon E5-2690 (Sandy Bridge) @ 2.90GHz
- 1x - 8x oversubscription (256 - 2048 MPI ranks, same problem size)

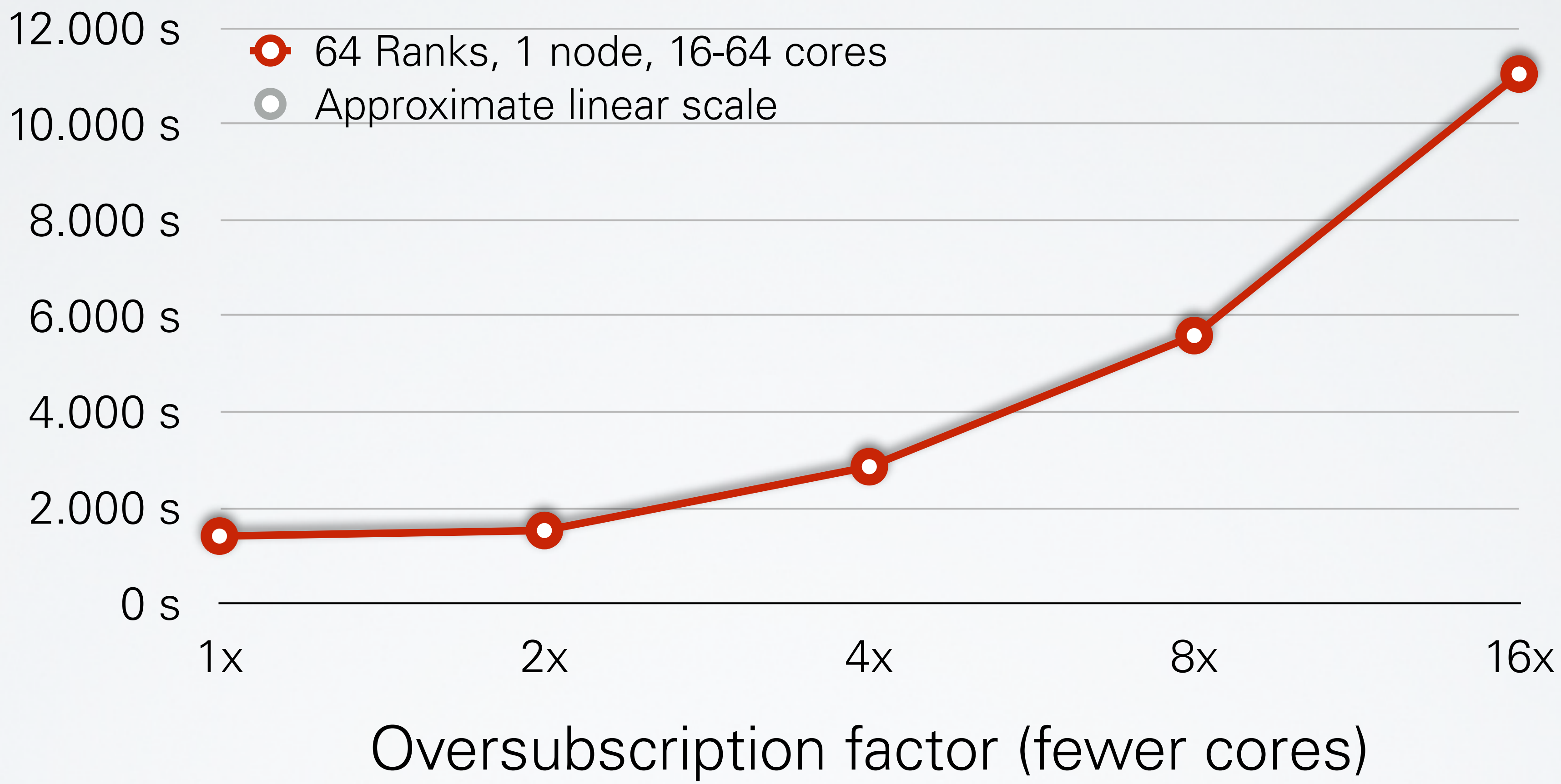
OVERSUBSCRIPTION



Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

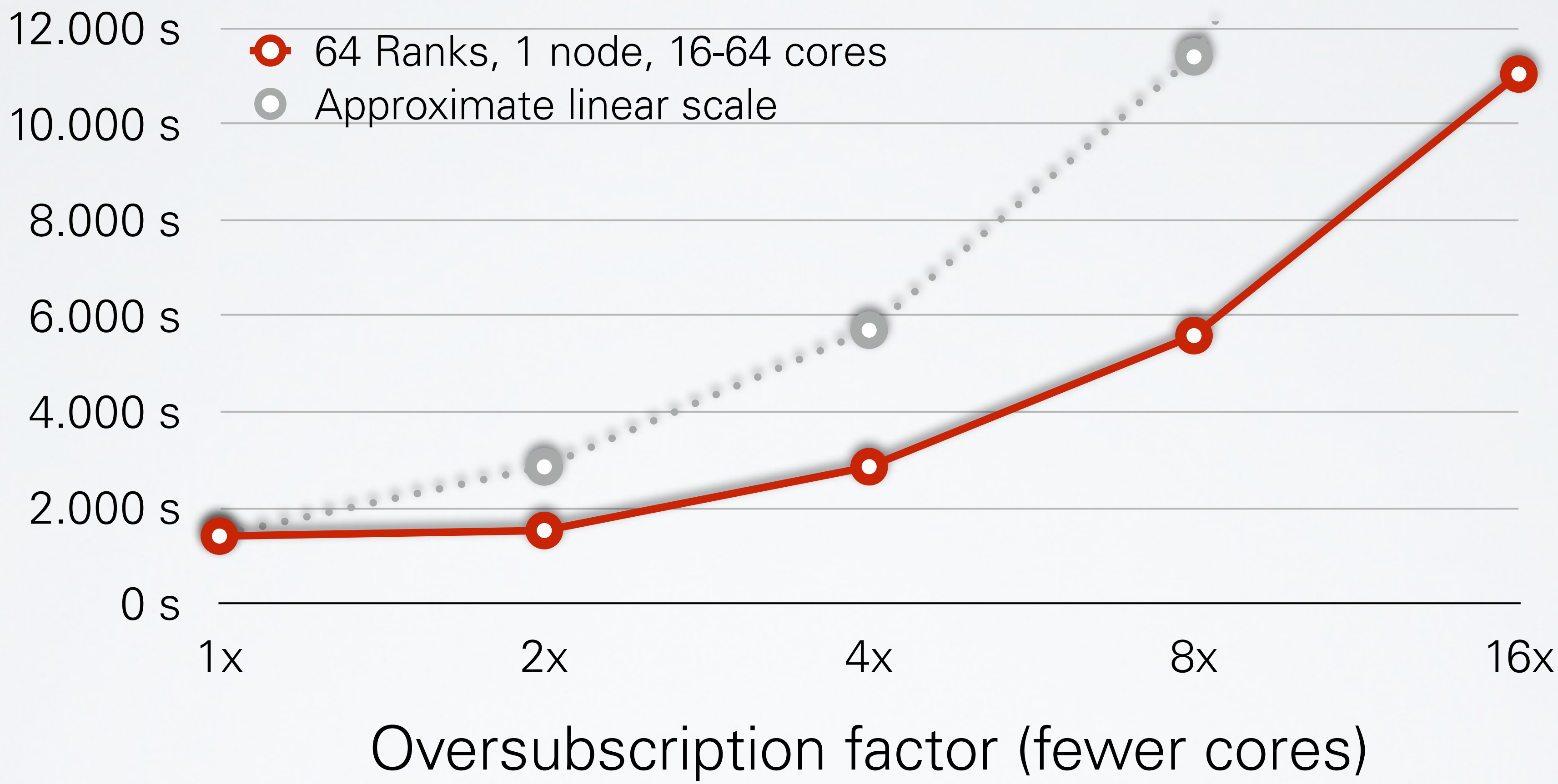
OVERSUBSCRIPTION



Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

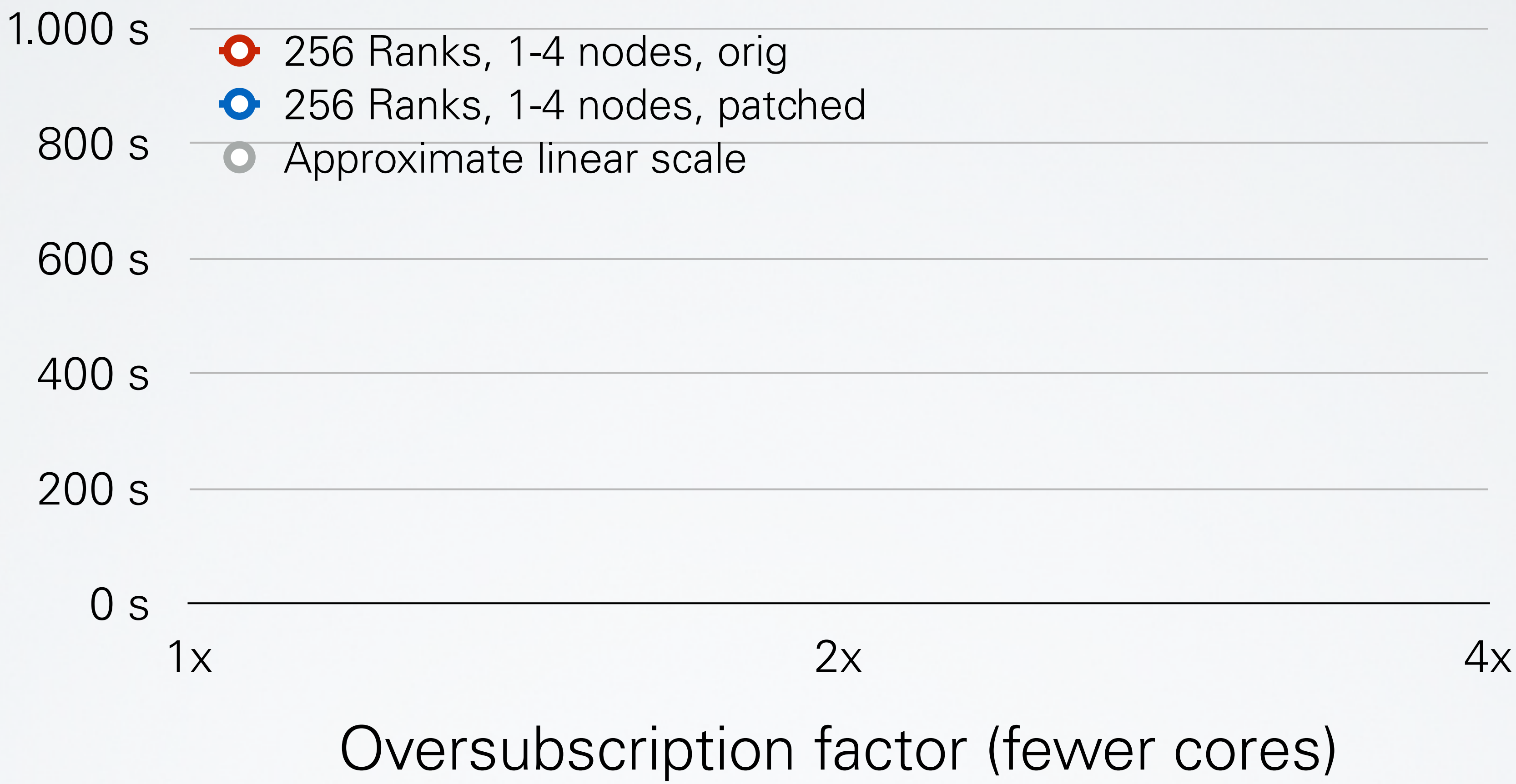
OVERSUBSCRIPTION



Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

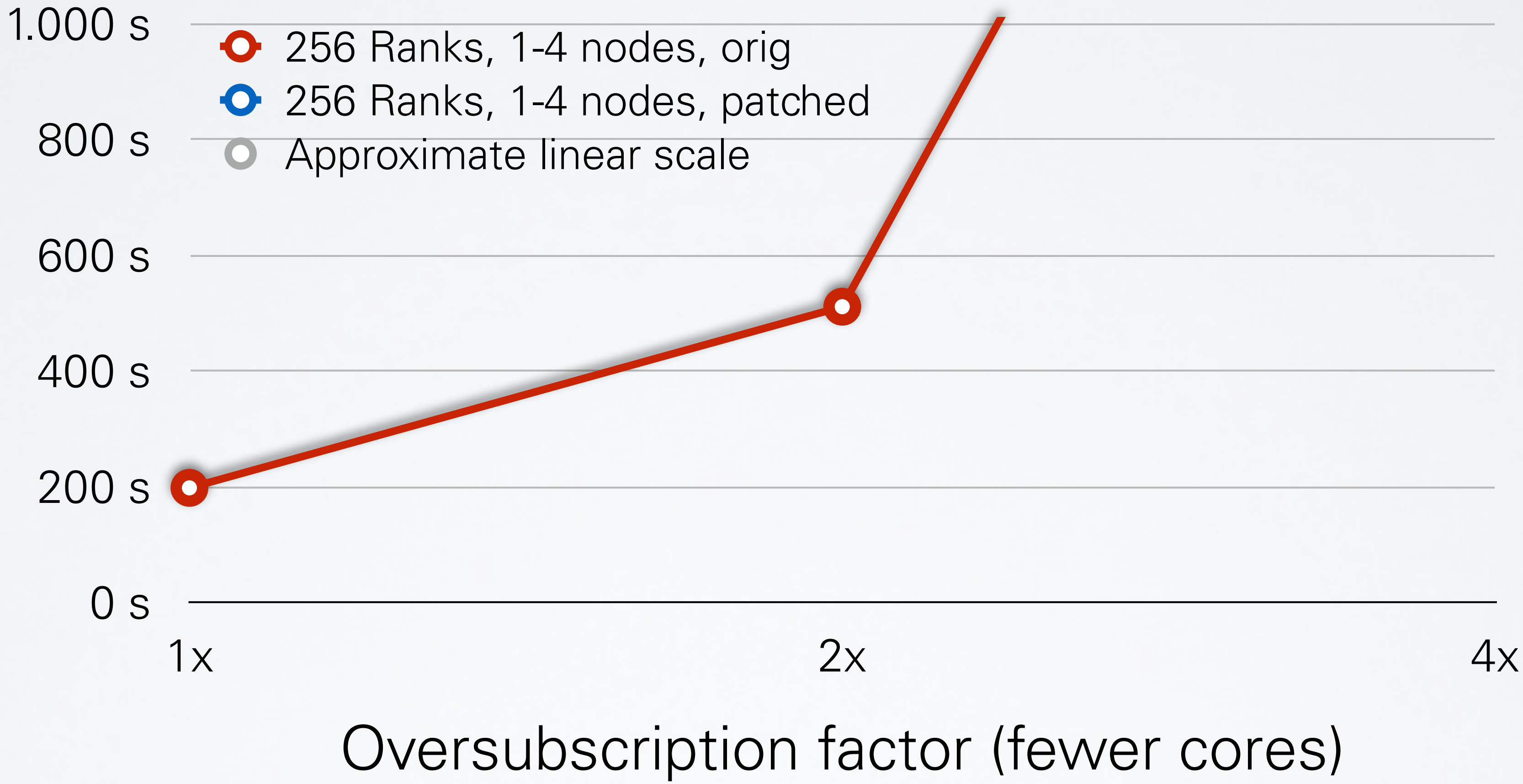
OVERSUBSCRIPTION



Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

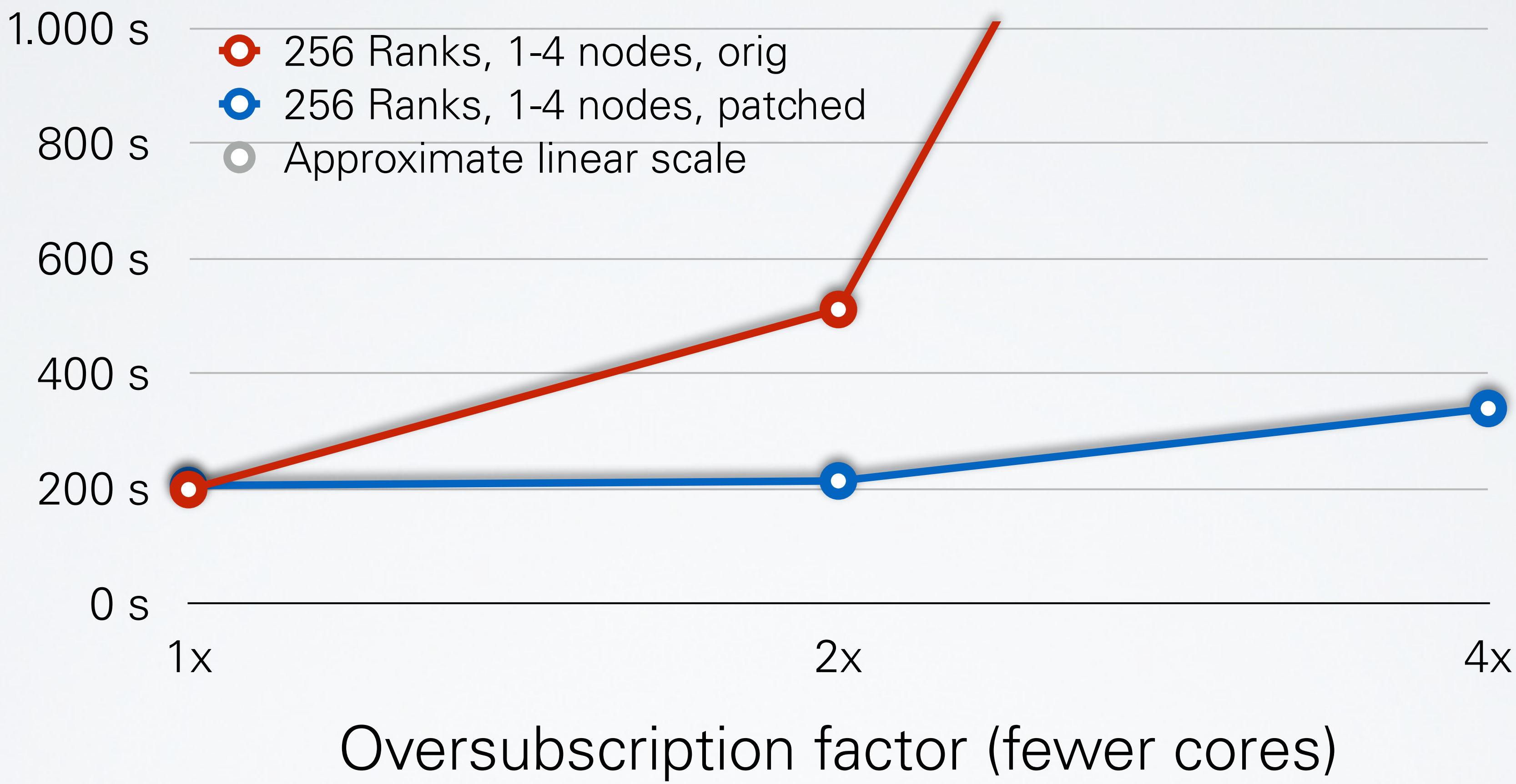
OVERSUBSCRIPTION



Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

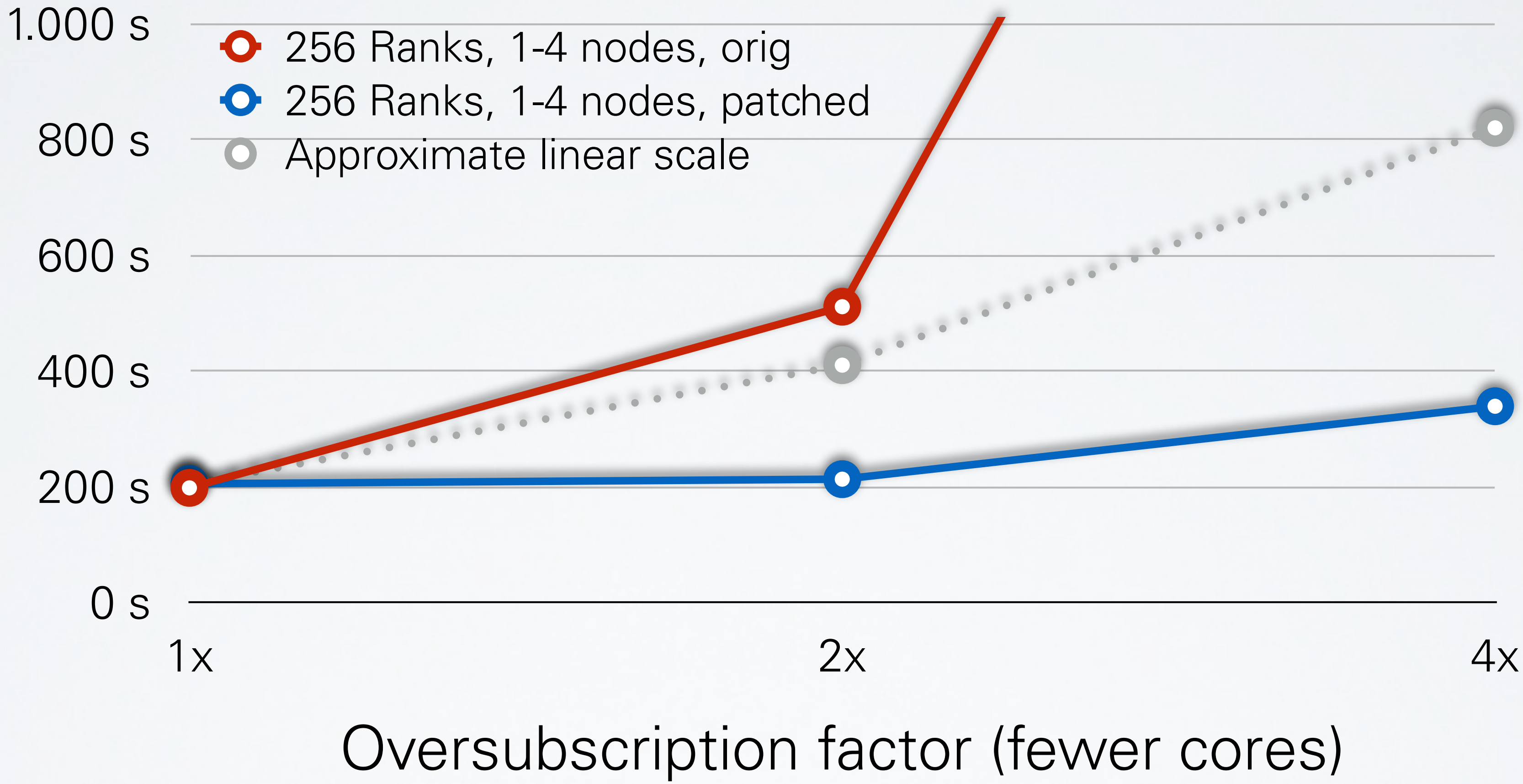
OVERSUBSCRIPTION



Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

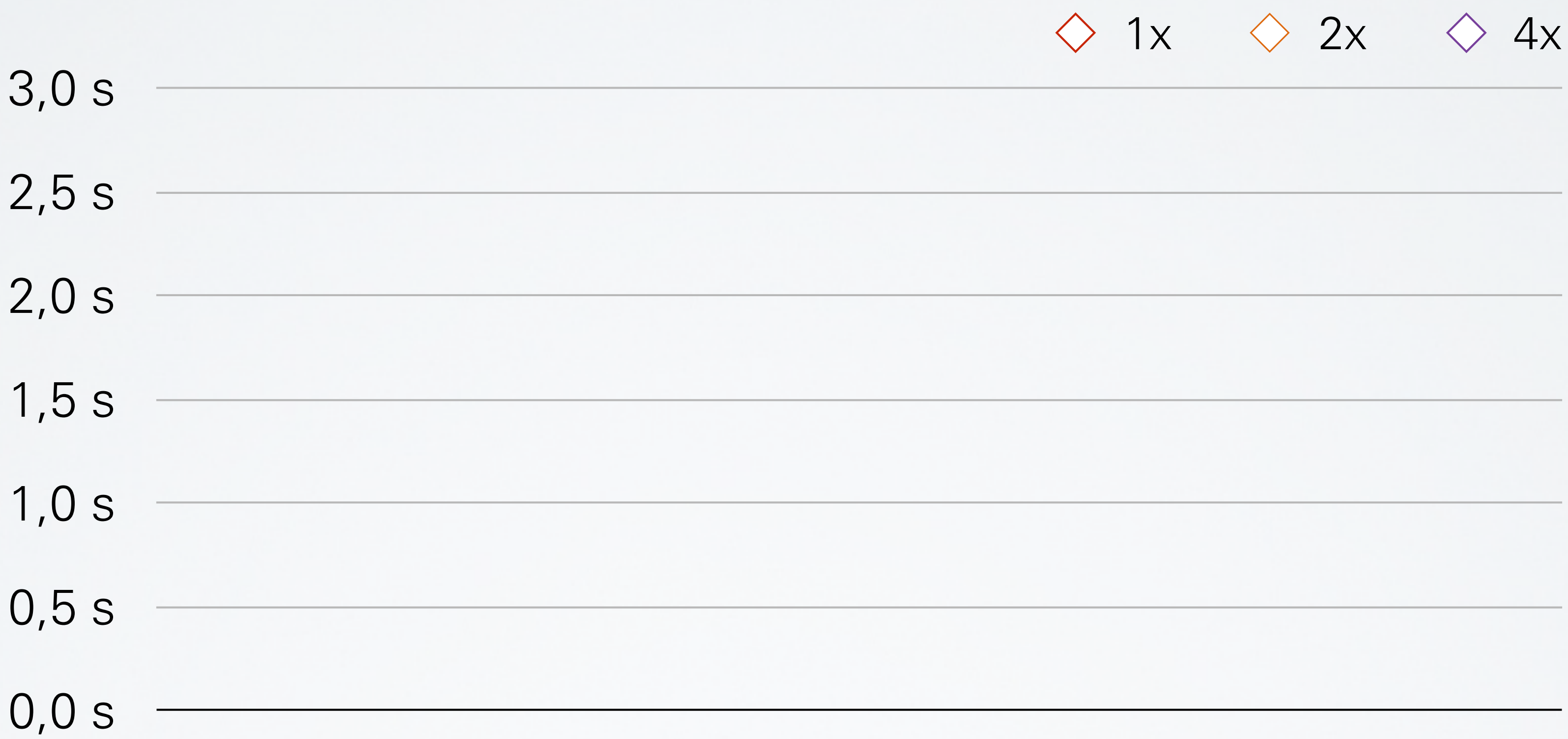
OVERSUBSCRIPTION



Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

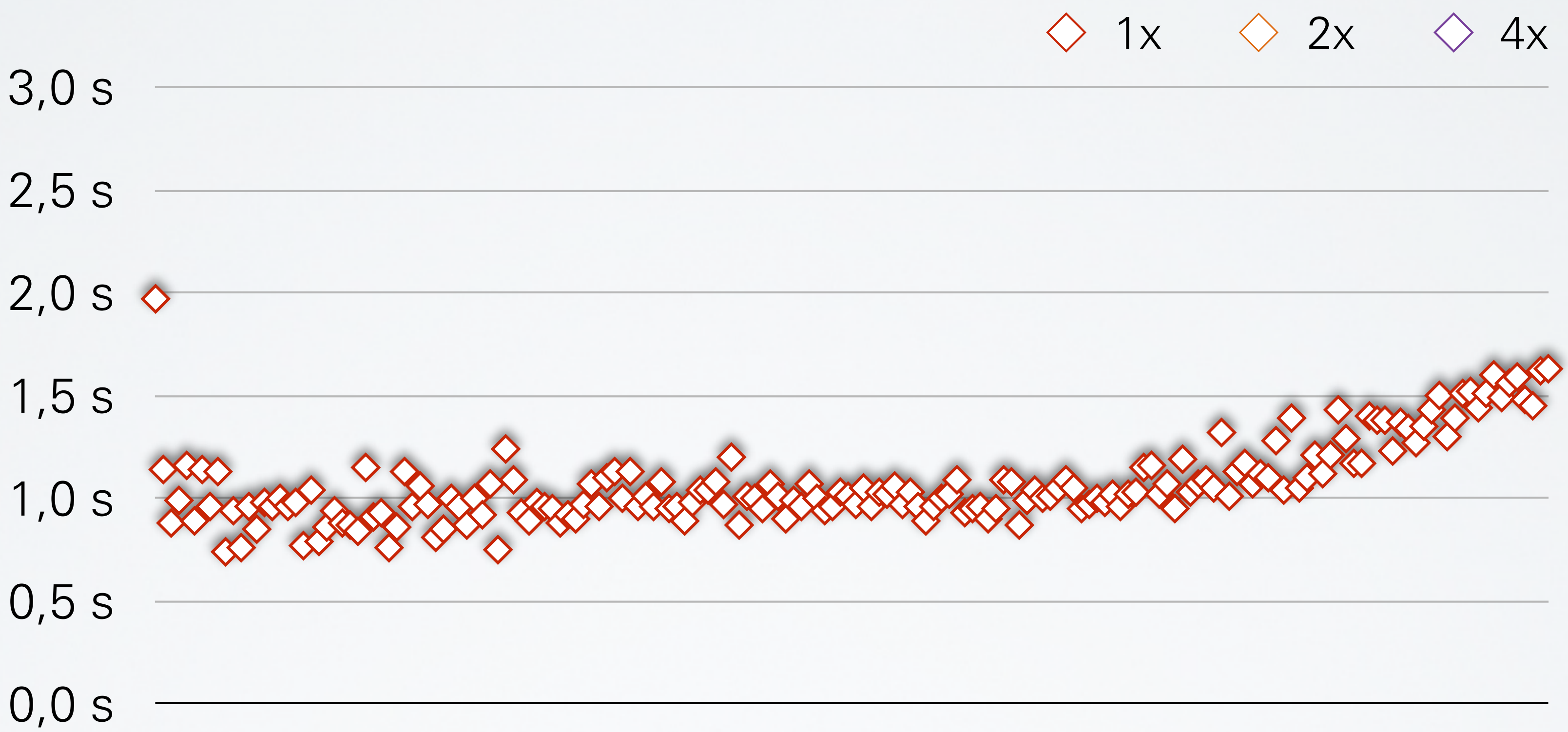
PATCHED: STEP TIME



Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

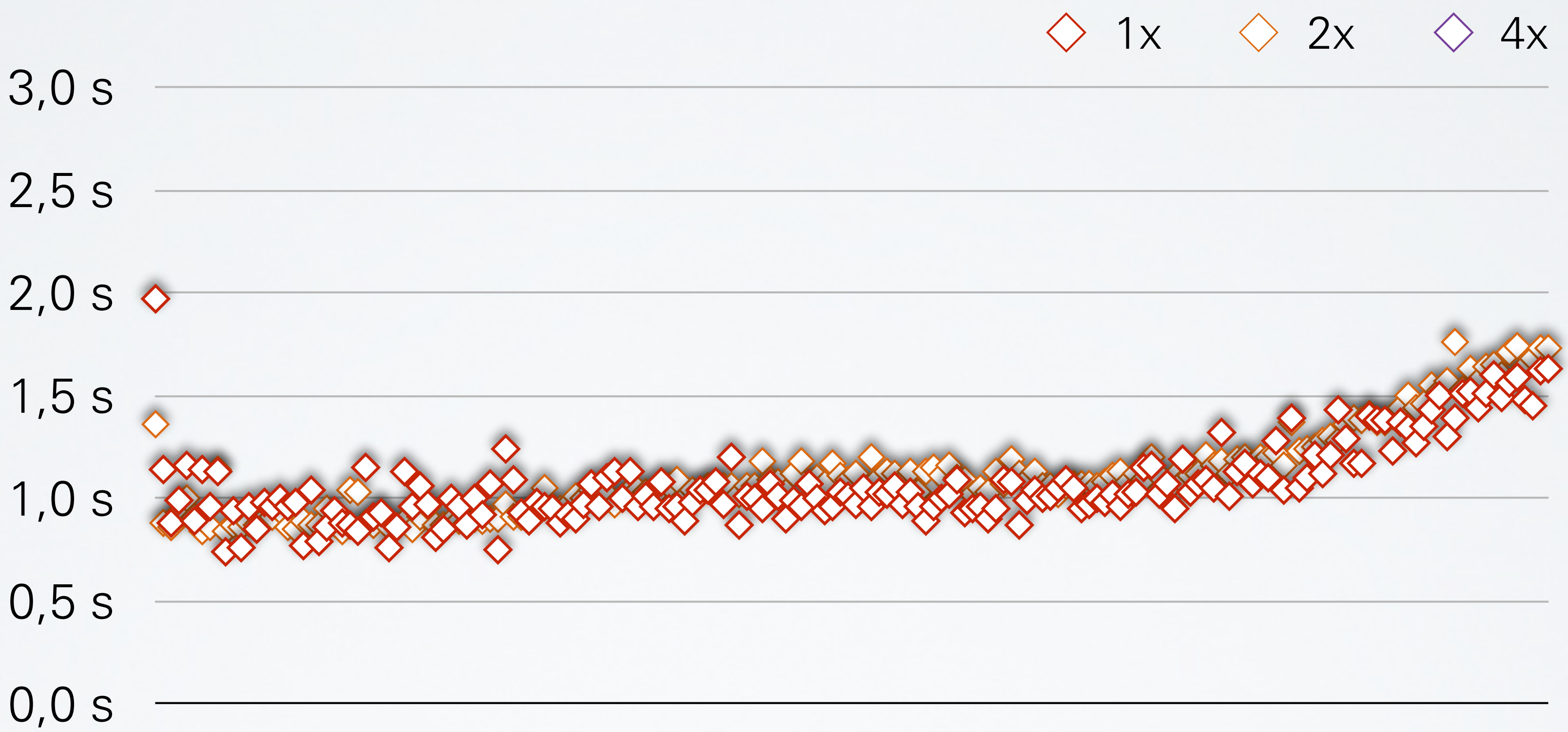
PATCHED: STEP TIME



Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

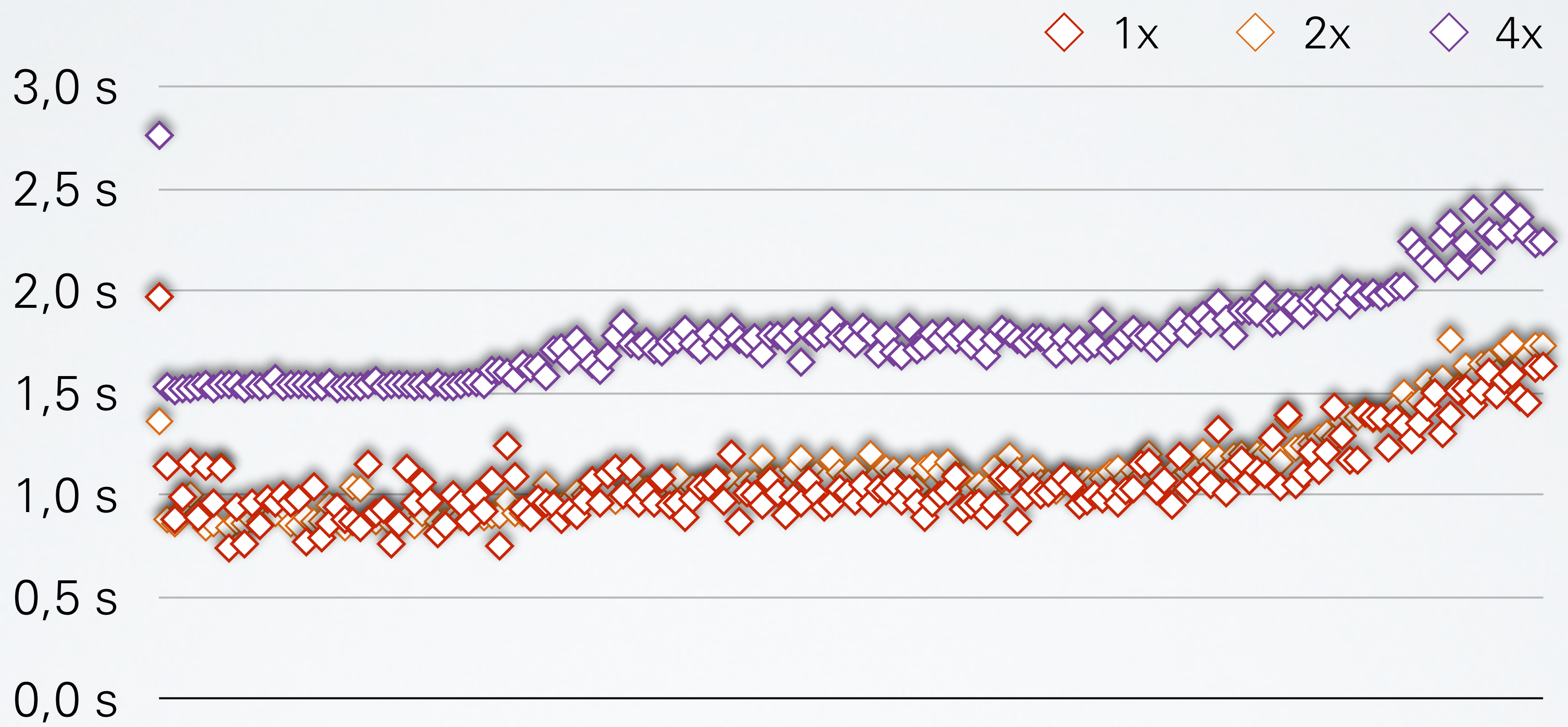
PATCHED: STEP TIME



Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

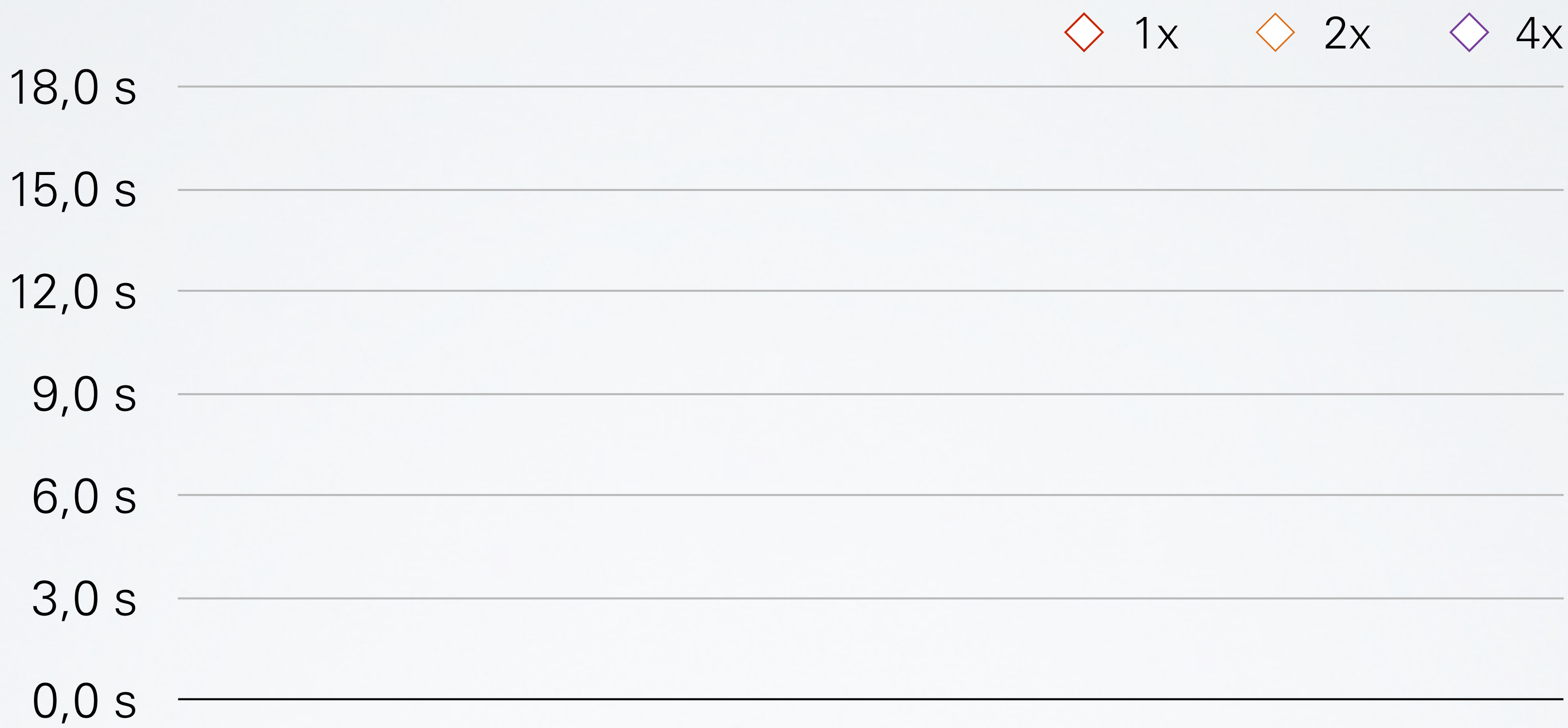
PATCHED: STEP TIME



Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

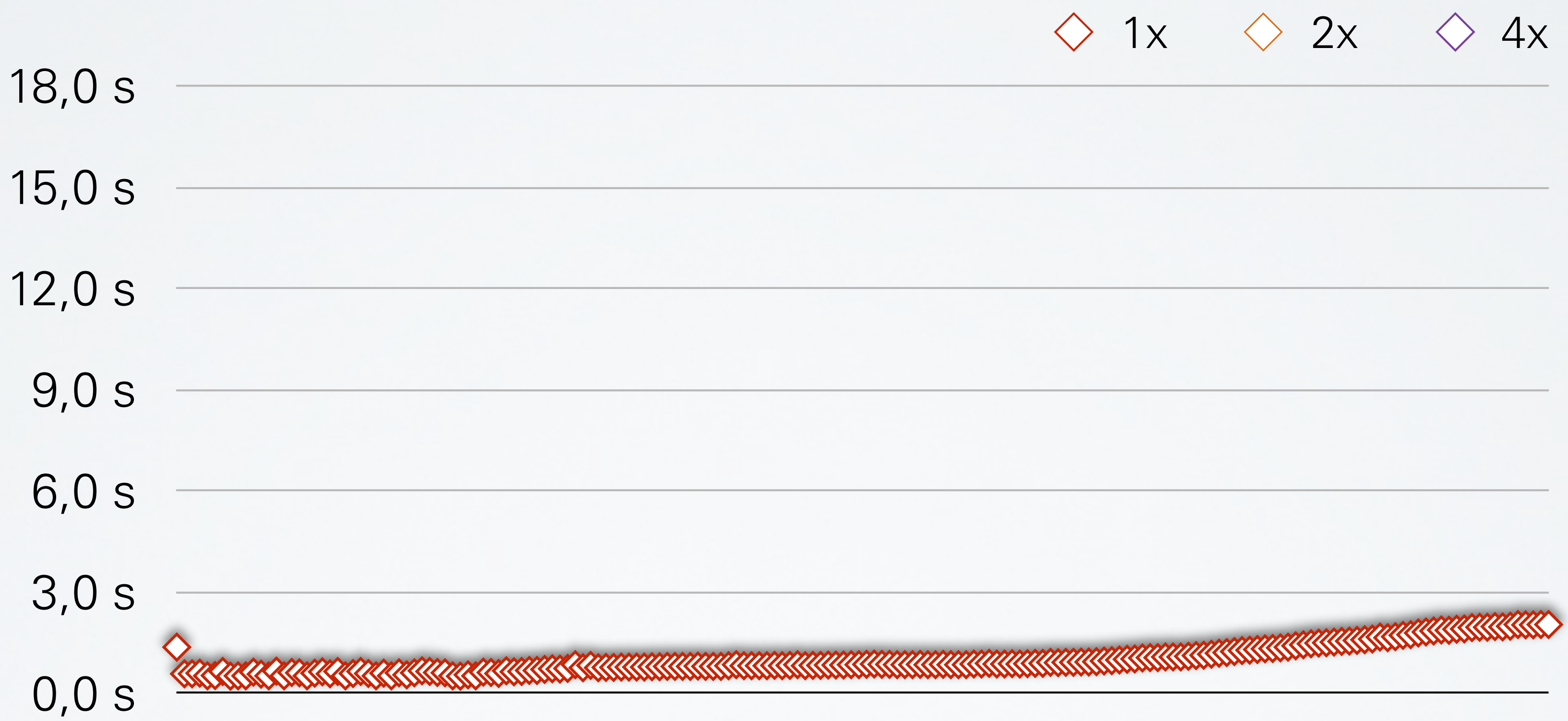
ORIG: STEP TIME



Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

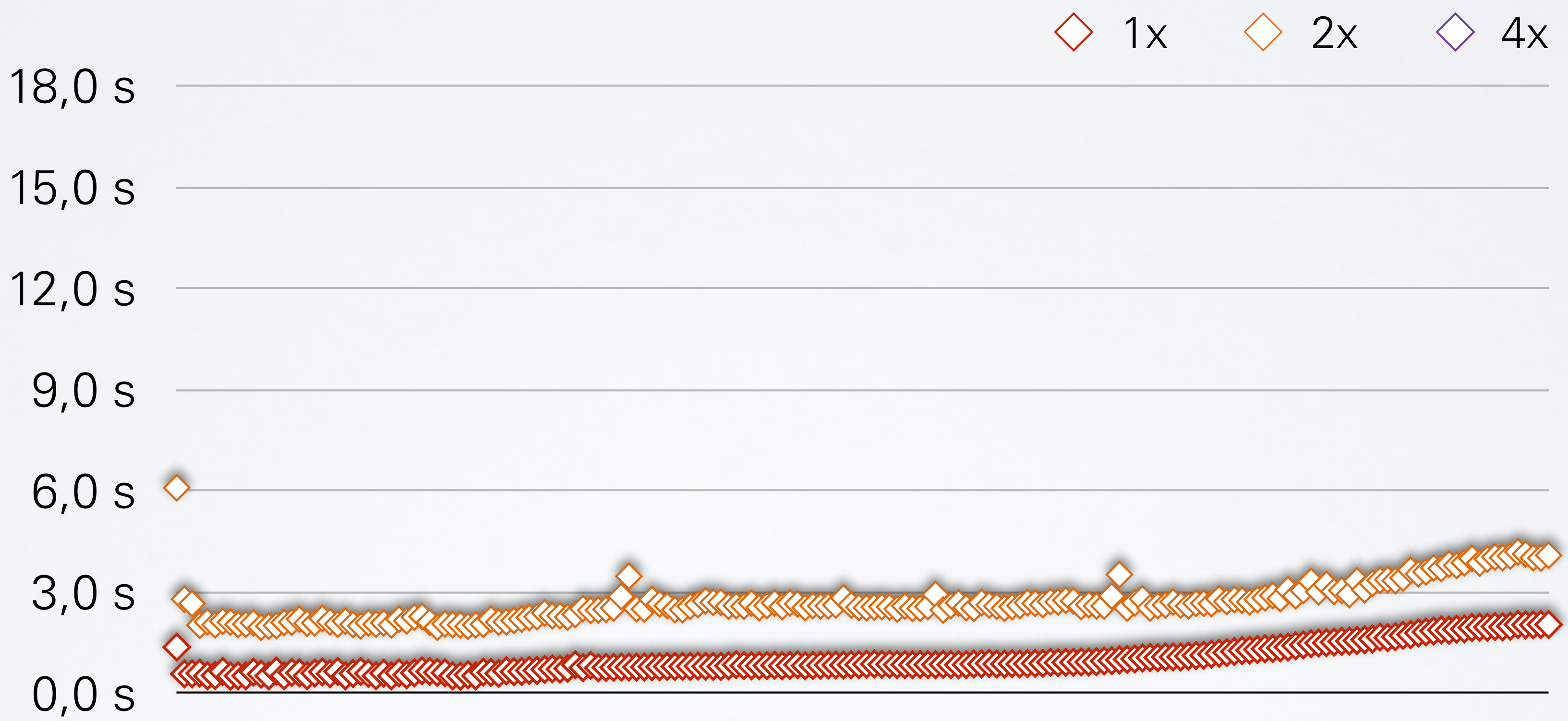
ORIG: STEP TIME



Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

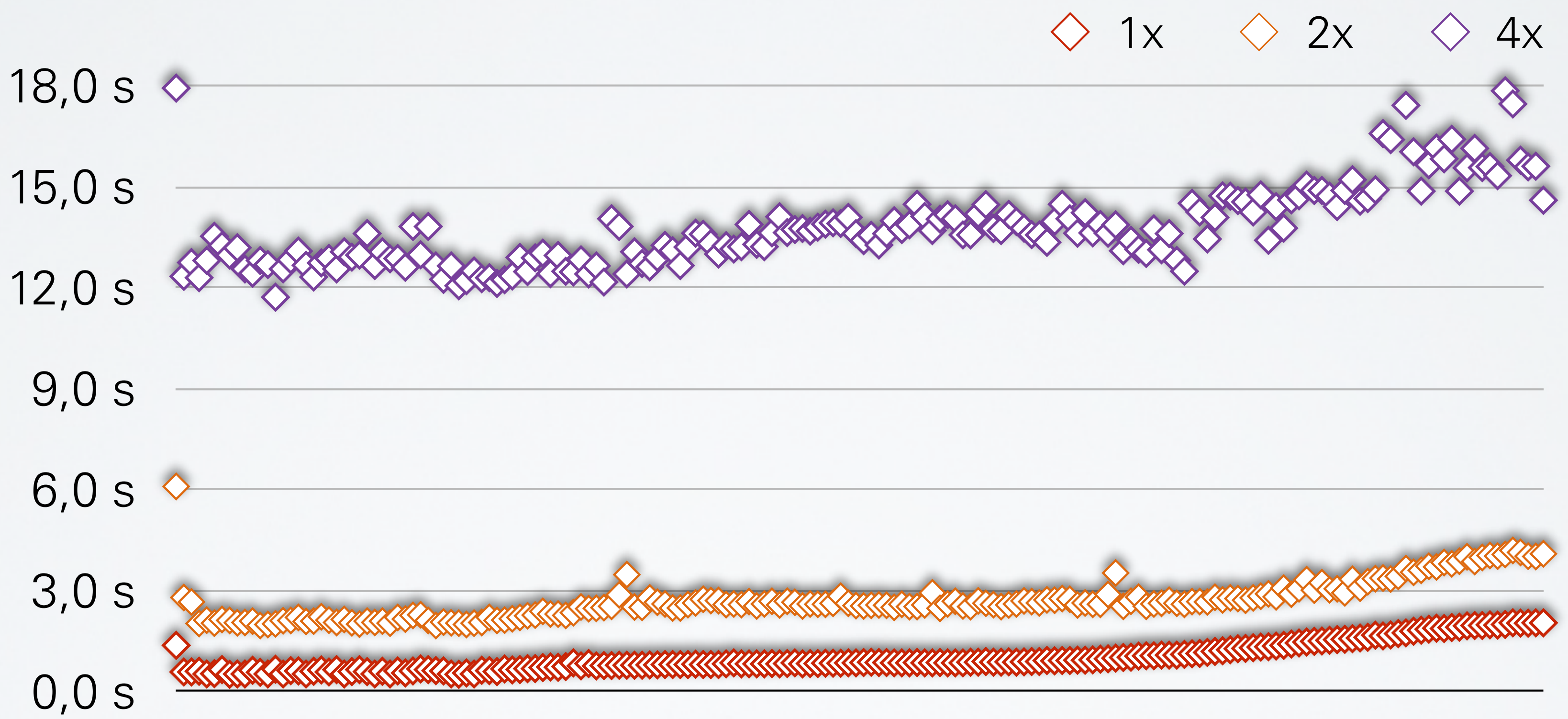
ORIG: STEP TIME



Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

ORIG: STEP TIME

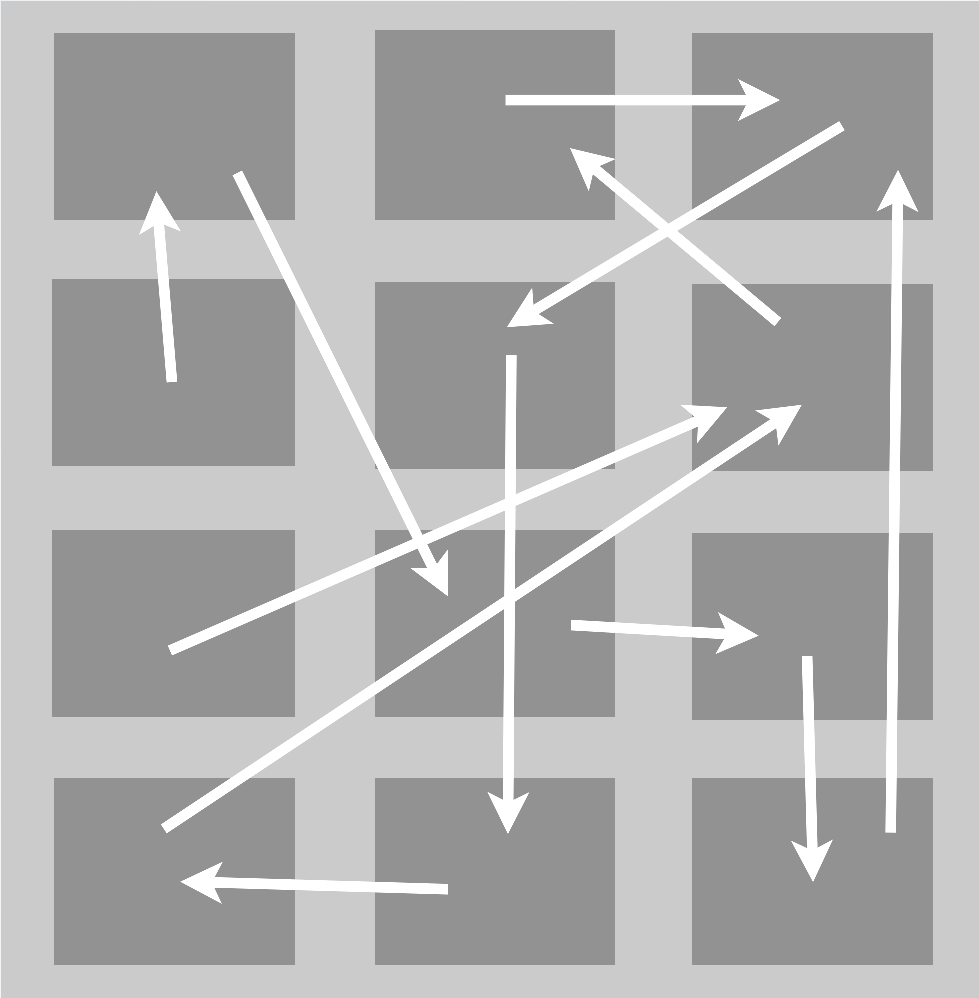


Application: COSMO-SPECS+FD4 (no load balancing)

- ATLAS nodes w/ 64 AMD Opteron 6274 cores @ 2.2 GHz
- Number of ranks remained constant, but number of cores was reduced

EXPERIMENTS: GOSSIP SCALABILITY

RANDOM GOSSIP



Distributed Bulletin Board

- Each node keeps vector with per-node info (own + info received from others)
- Once per time step, each node sends to 1 other randomly selected node a subset of its own vector entries (called "window")
- Node merges received window entries into local vector (if newer)

MOSIX: GOSSIP ALGORITHM

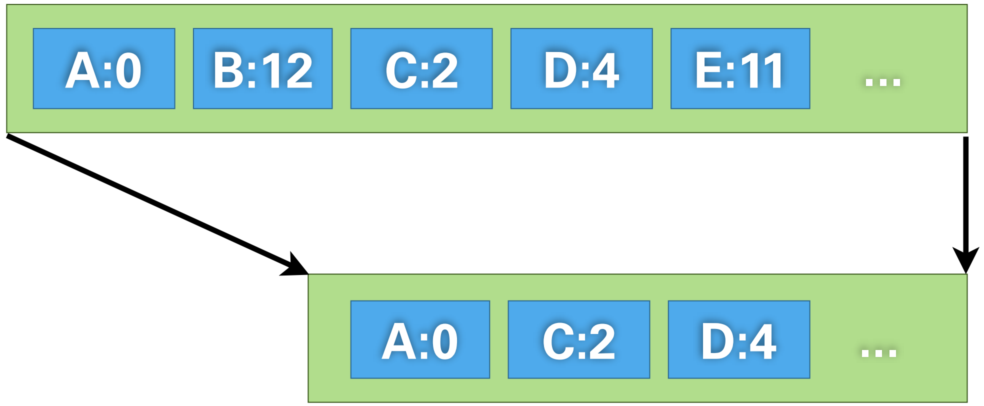
Each time unit:

- Update local info
- Find all vector entries up to age T (called a window)
- Send window to 1 randomly selected node

MOSIX: GOSSIP ALGORITHM

Each time unit:

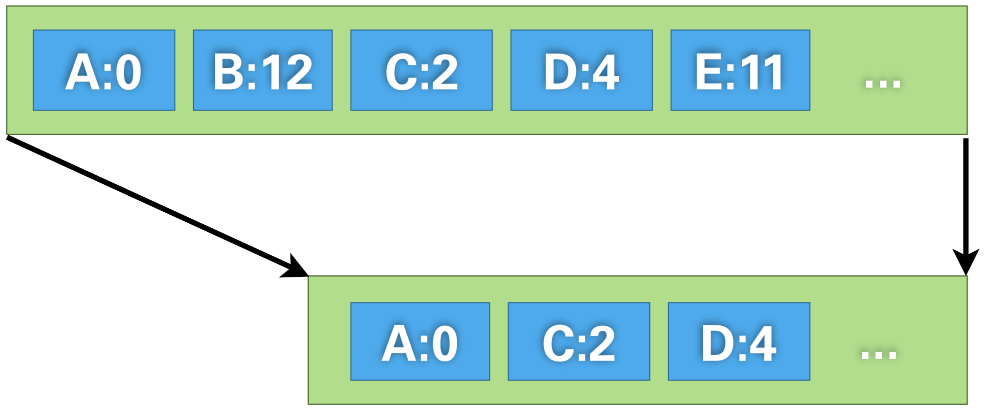
- Update local info
- Find all vector entries up to age T (called a window)
- Send window to 1 randomly selected node



MOSIX: GOSSIP ALGORITHM

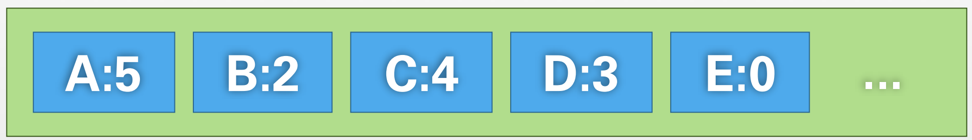
Each time unit:

- Update local info
- Find all vector entries up to age T (called a window)
- Send window to 1 randomly selected node



Upon receiving a window:

- Update the received entries' age (+1 for transfer)
- Update entries in local vector where newer information has been received



MOSIX: GOSSIP ALGORITHM

Each time unit:

- Update local info
- Find all vector entries up to age T (called a window)
- Send window to 1 randomly selected node

A:0 | B:12 | C:2 | D:4 | E:11 | ...

Upon receiving a window:

- Update the received entries' age (+1 for transfer)
- Update entries in local vector where newer information has been received

A:0 | C:2 | D:4 | ...

A:5 | B:2 | C:4 | D:3 | E:0 | ...

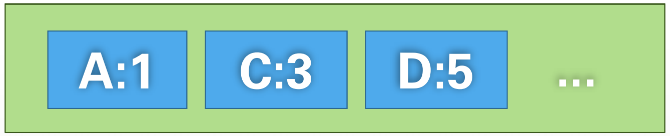
MOSIX: GOSSIP ALGORITHM

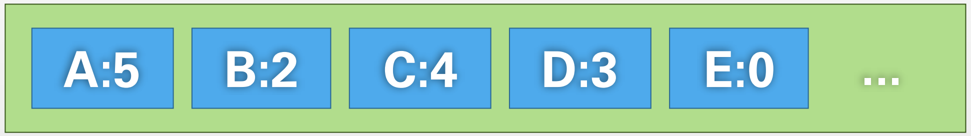
Each time unit:

- Update local info
- Find all vector entries up to age T (called a window)
- Send window to 1 randomly selected node

Upon receiving a window:

- Update the received entries' age (+1 for transfer)
- Update entries in local vector where newer information has been received





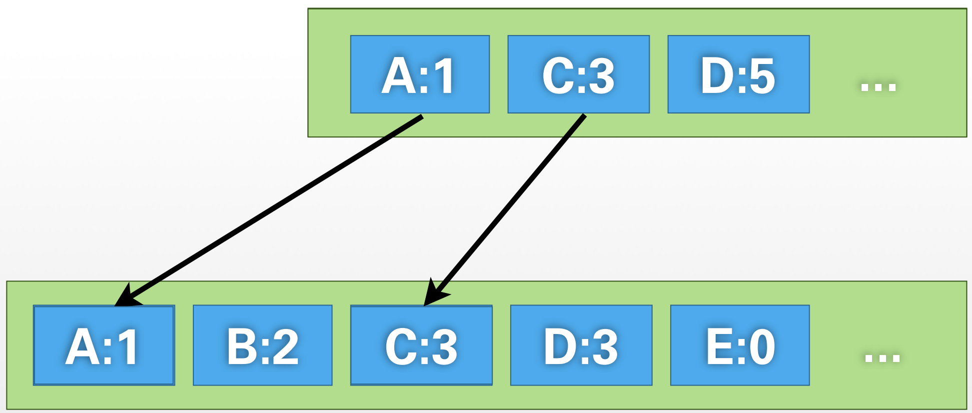
MOSIX: GOSSIP ALGORITHM

Each time unit:

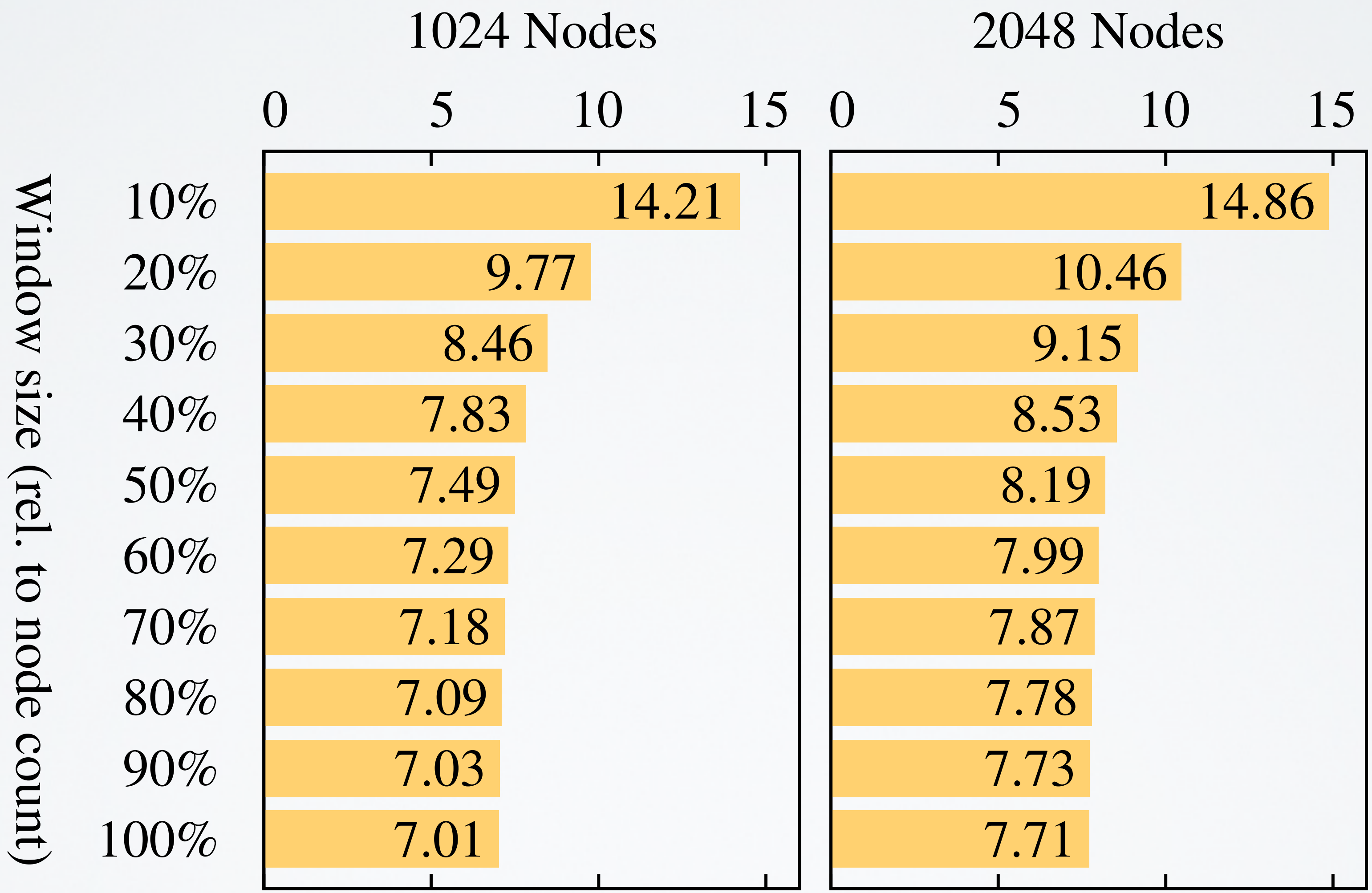
- Update local info
- Find all vector entries up to age T (called a window)
- Send window to 1 randomly selected node

Upon receiving a window:

- Update the received entries' age (+1 for transfer)
- Update entries in local vector where newer information has been received



WINDOW SIZE


1024 Nodes
2048 Nodes
0
5
10
15
Window size (rel. to node count)
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
14.21
9.77
8.46
7.83
7.49
7.29
7.18
7.09
7.03
7.01
14.86
10.46
9.15
8.53
8.19
7.99
7.87
7.78
7.73
7.71

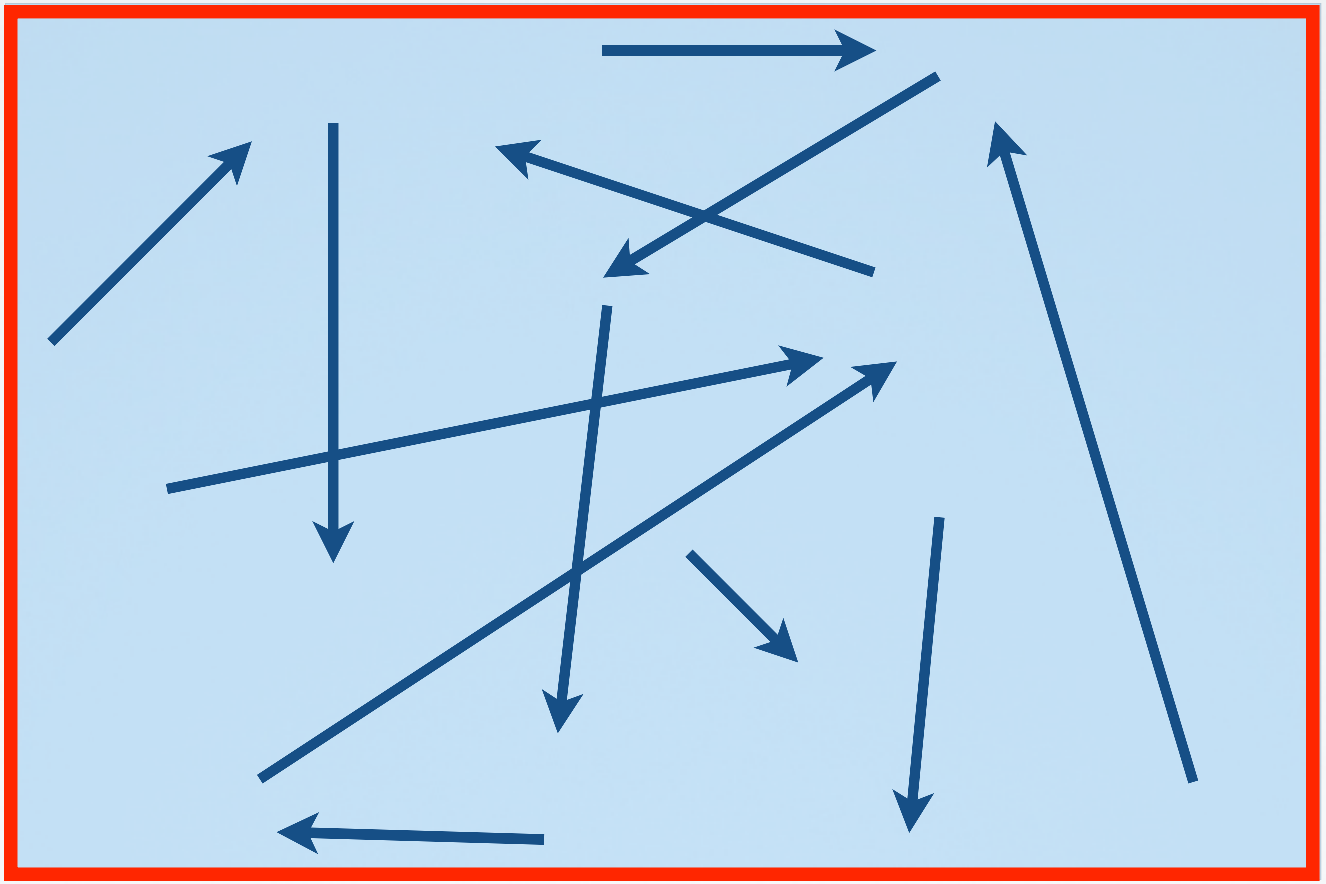
NODES: VECTOR AGE

Colony nodes	Method	Circulating among colony nodes: windows not exceeding age 2	4	6	8	10	whole vector
128	Approx.	19.15	6.00	4.93	4.89	4.89	4.89
	Simulation	18.87	6.04	4.97	4.92	4.95	
	Measured	18.75	5.99	4.94	4.88	4.90	
256	Approx.	36.49	8.49	5.70	5.57	5.57	5.57
	Simulation	36.33	8.57	5.77	5.63	5.62	
	Measured	36.06	8.55	5.77	5.60	5.60	
512	Approx.	71.15	13.27	6.70	6.26	6.25	6.25
	Simulation	71.01	13.34	6.81	6.34	6.32	
	Measured	70.85	13.37	6.78	6.31	6.28	
1K	Approx.	140.44	22.69	8.21	6.99	6.94	6.94
	Simulation	139.76	22.73	8.33	7.06	7.01	
	Measured	140.14	22.83	8.32	7.04	6.98	
2K	Approx.	279.03	41.47	10.90	7.79	7.63	7.63
	Simulation	267.82	41.58	11.08	7.89	7.71	
	Measured	278.94	41.66	11.03	7.84	7.66	
4K	Approx.	556.20	78.99	16.06	8.83	8.34	8.32
	Simulation	479.96	79.10	16.23	8.95	8.42	
	Measured	556.20	79.39	16.24	8.87	8.33	
8K	Approx.	1,110.53	154.02	26.26	10.44	9.07	9.01
	Simulation	798.97	153.80	26.48	10.59	9.43	
	Measured	1,102.99	155.16	26.51	10.44	8.98	
1M	Approx.	141,911	19,209	2,605	360	58	13.86
1G	Approx.	145M	19M	2M	360K	48K	20.79

NODES: VECTOR AGE

Colony nodes	Method	Circulating among colony nodes: windows not exceeding age 2	4	6	8	10	whole vector
128	Approx.	19.15	6.00	4.93	4.89	4.89	4.89
	Simulation	18.87	6.04	4.97	4.92	4.95	
	Measured	18.75	5.99	4.94	4.88	4.90	
256	Approx.	36.49	8.49	5.70	5.57	5.57	5.57
	Simulation	36.33	8.57	5.77	5.63	5.62	
	Measured	36.06	8.55	5.77	5.60	5.60	
512	Approx.	71.15	13.27	6.70	6.26	6.25	6.25
	Simulation	71.01	13.34	6.81	6.34	6.32	
	Measured	70.85	13.37	6.78	6.31	6.28	
1K	Approx.	140.44	22.69	8.21	6.99	6.94	6.94
	Simulation	139.76	22.73	8.33	7.06	7.01	
	Measured	140.14	22.83	8.32	7.04	6.98	
2K	Approx.	279.03	41.47	10.90	7.79	7.63	7.63
	Simulation	267.82	41.58	11.08	7.89	7.71	
	Measured	278.94	41.66	11.03	7.84	7.66	
4K	Approx.	556.20	78.99	16.06	8.83	8.34	8.32
	Simulation	479.96	79.10	16.23	8.95	8.42	
	Measured	556.20	79.39	16.24	8.87	8.33	
8K	Approx.	1,110.53	154.02	26.26	10.44	9.07	9.01
	Simulation	798.97	153.80	26.48	10.59	9.43	
	Measured	1,102.99	155.16	26.51	10.44	8.98	
1M	Approx.	141,911	19,209	2,605	360	58	13.86
1G	Approx.	145M	19M	2M	360K	48K	20.79

SCALABILITY LIMITS

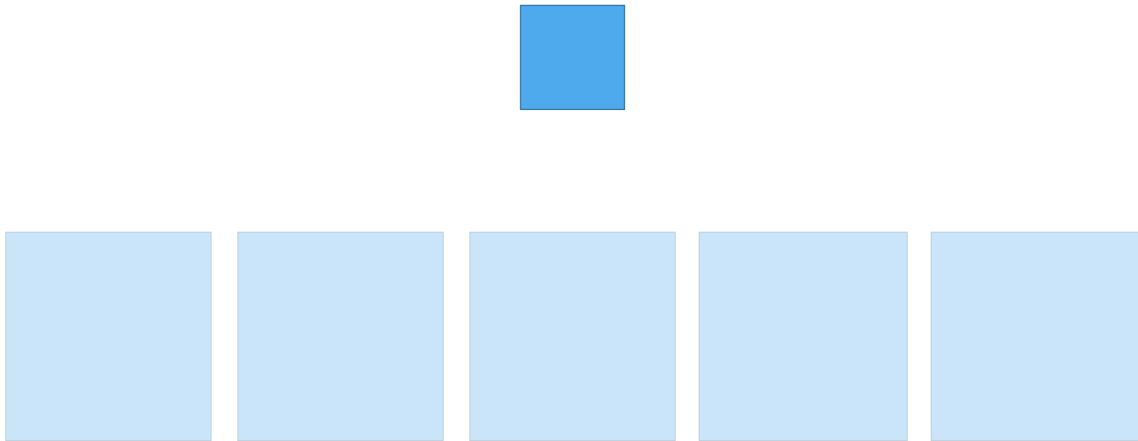


Problem: average age or window sizes too big for extreme numbers of nodes

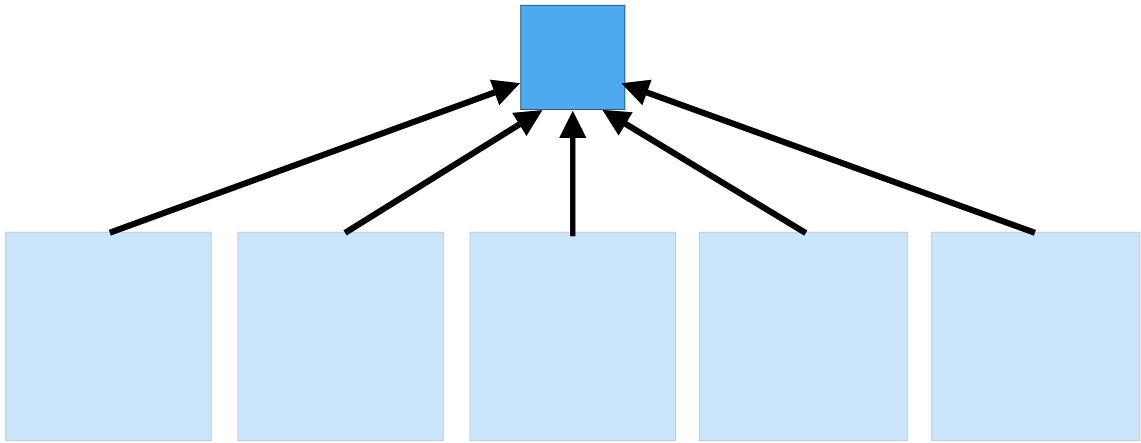
MASTER: GLOBAL VIEW

MASTER: GLOBAL VIEW

MASTER: GLOBAL VIEW

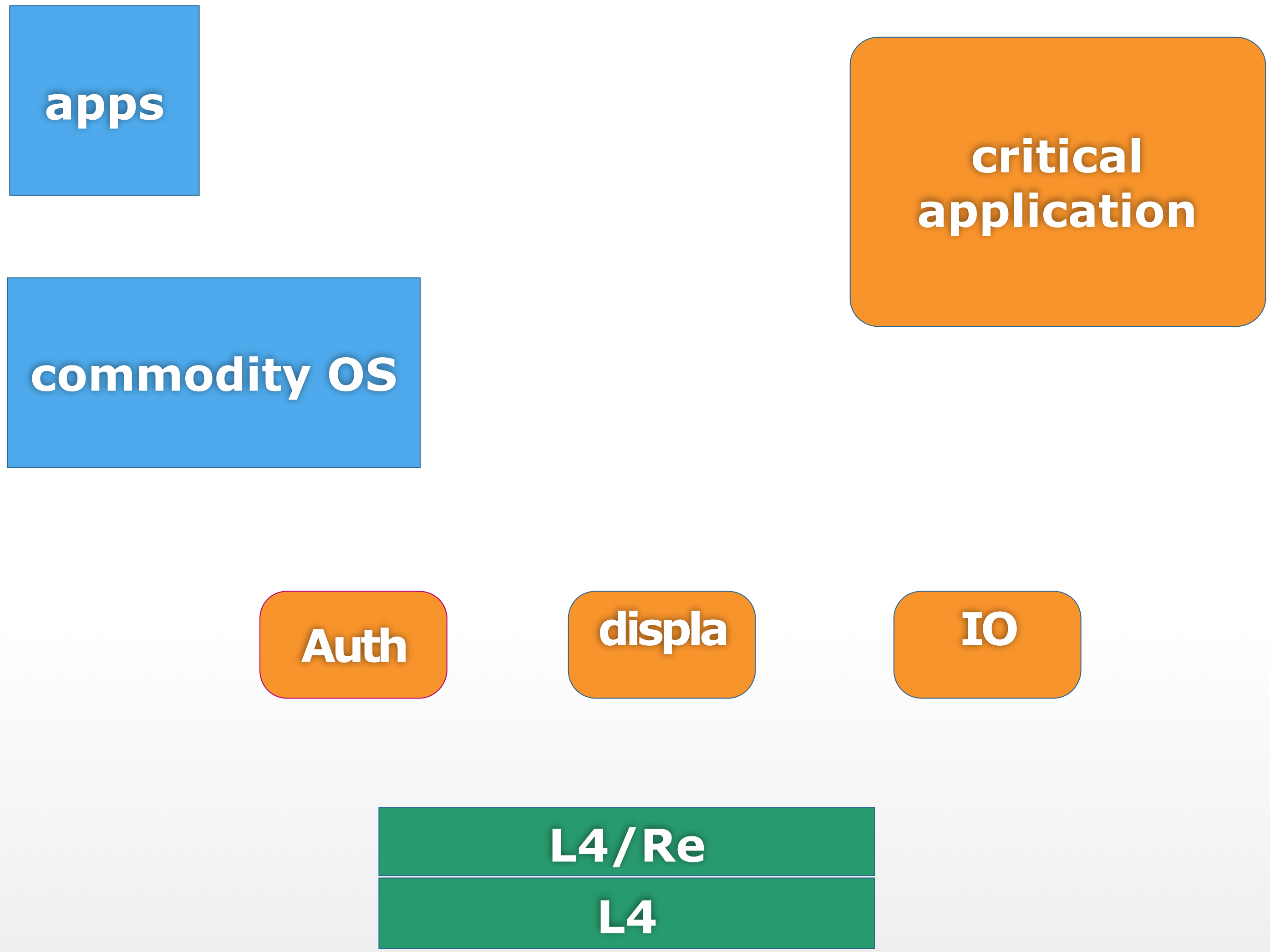


MASTER: GLOBAL VIEW



SYSTEM ARCHITECTURE

L4 MICRO KERNELS



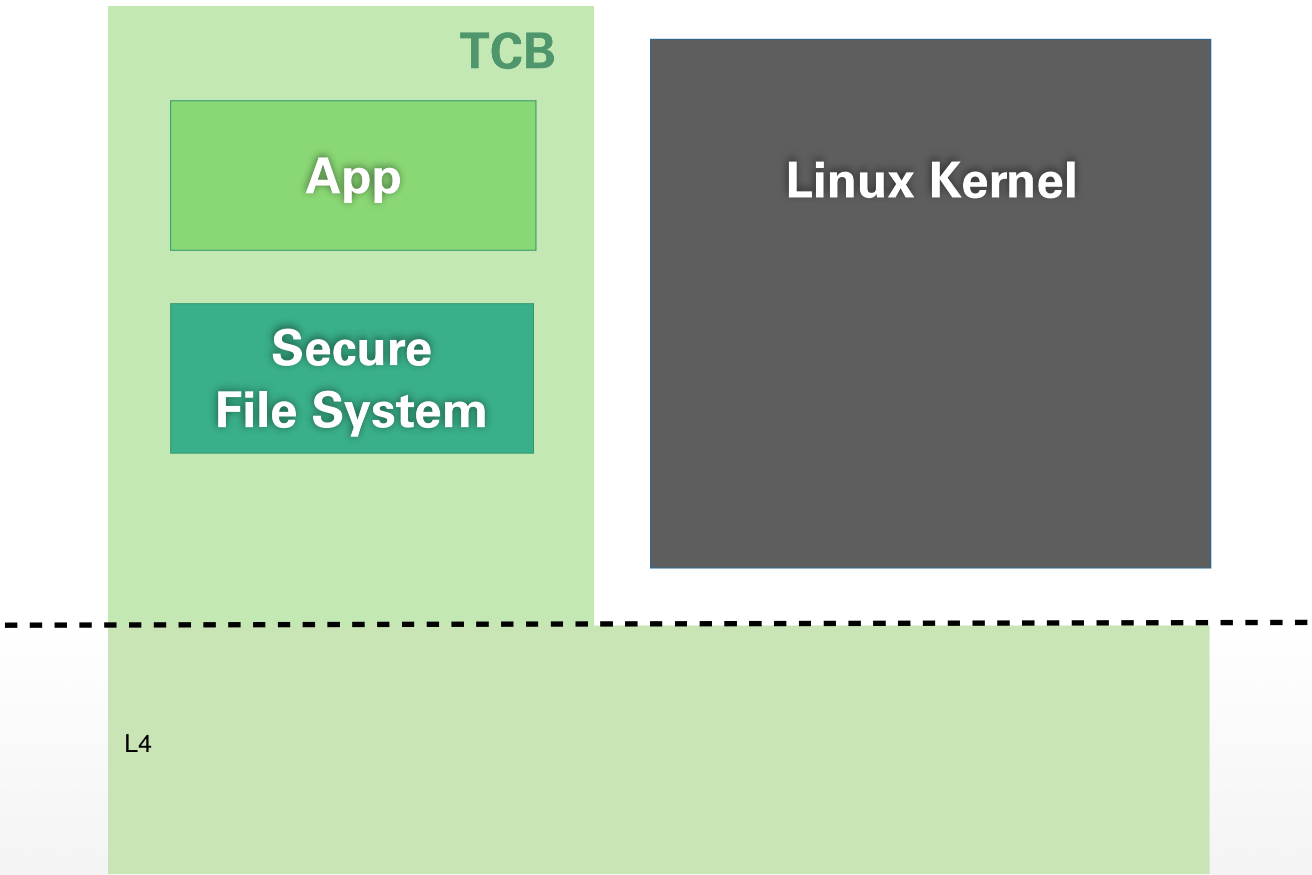
SIMKO 3

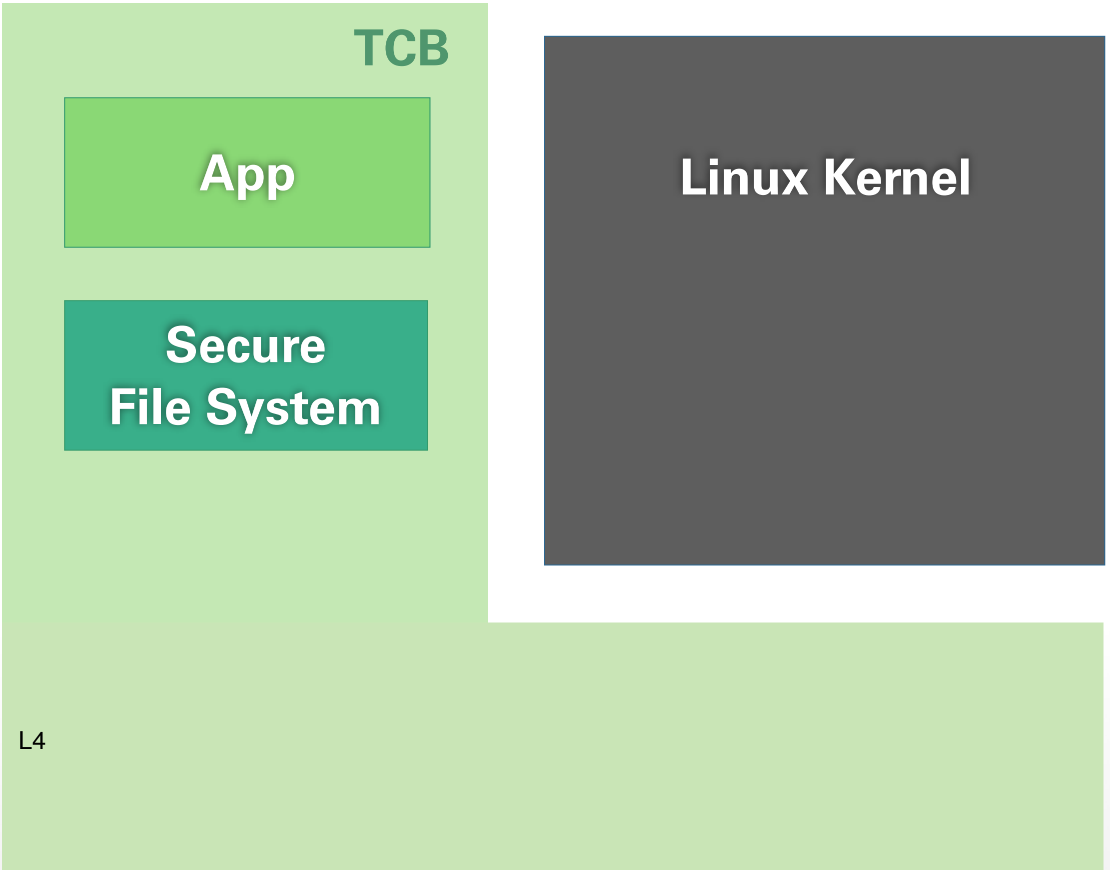

SAMSUNG
16:01
simko3
Bitte PIN eingeben
1
2
3
4
5
6
7
8
9
OK
0
Abbrechen

TECHNISCHE UNIVERSITÄT DRESDEN

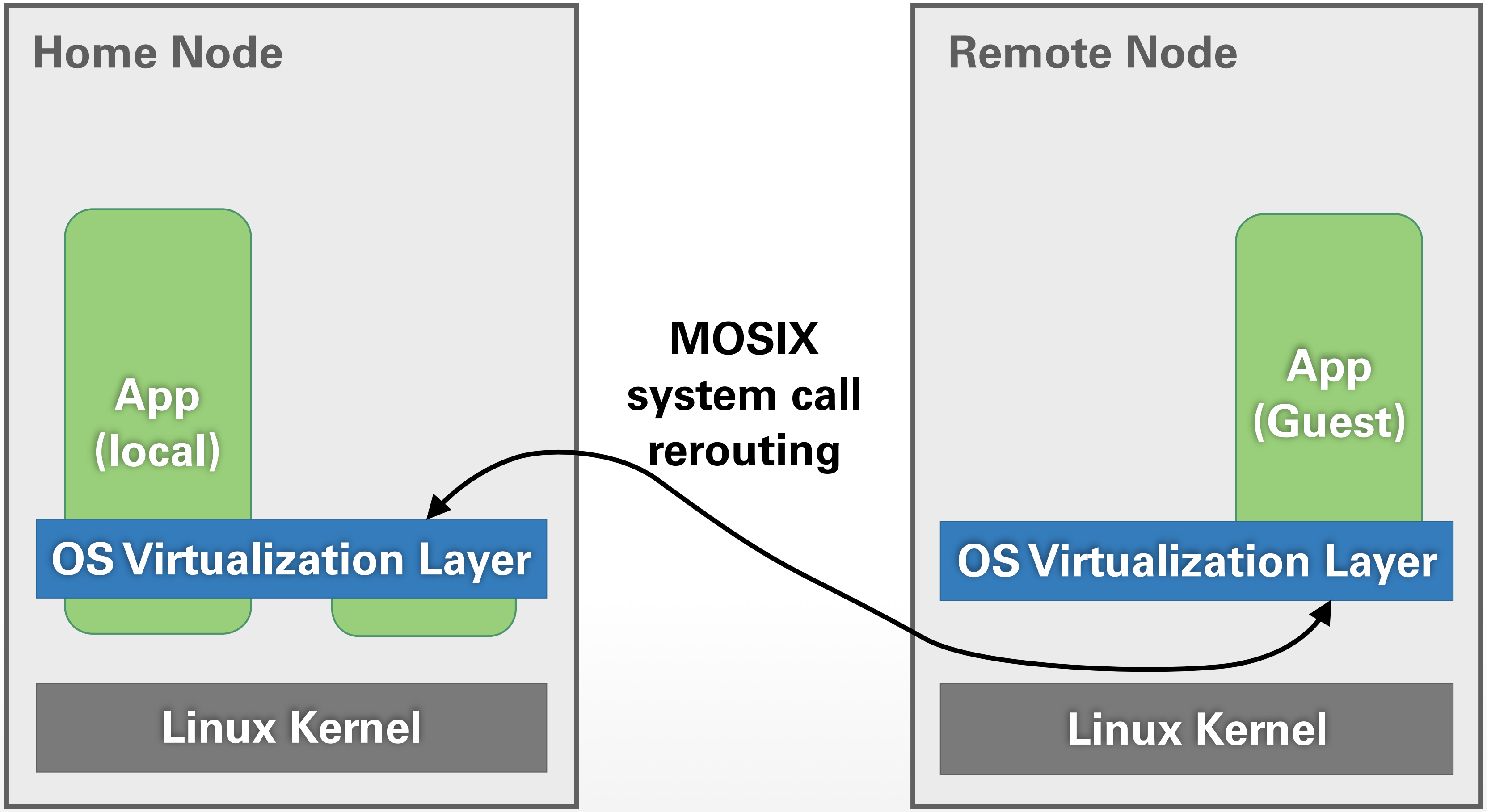
SIMKO 3

“ Merkel Phone “

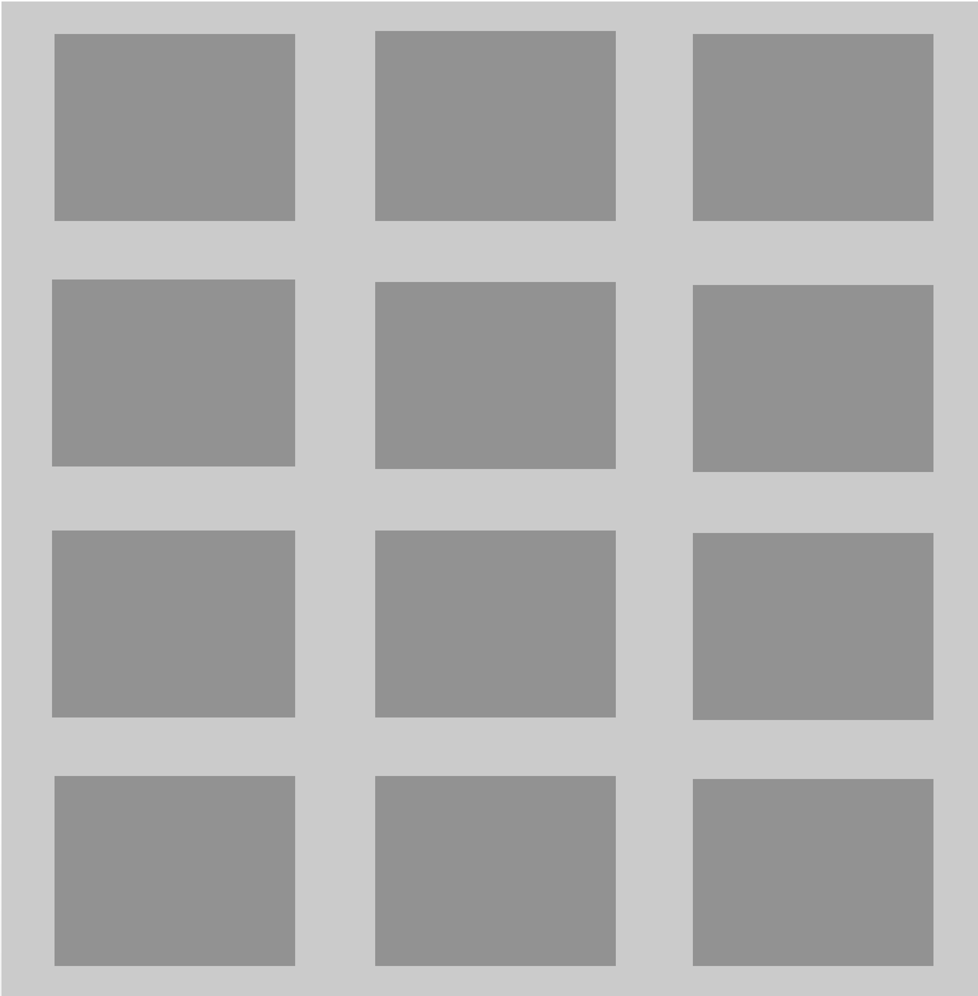

TCB
App
Secure File System
Linux Kernel
L4


TCB
App
Secure
File System
Linux Kernel
L4

MOSIX MIGRATION



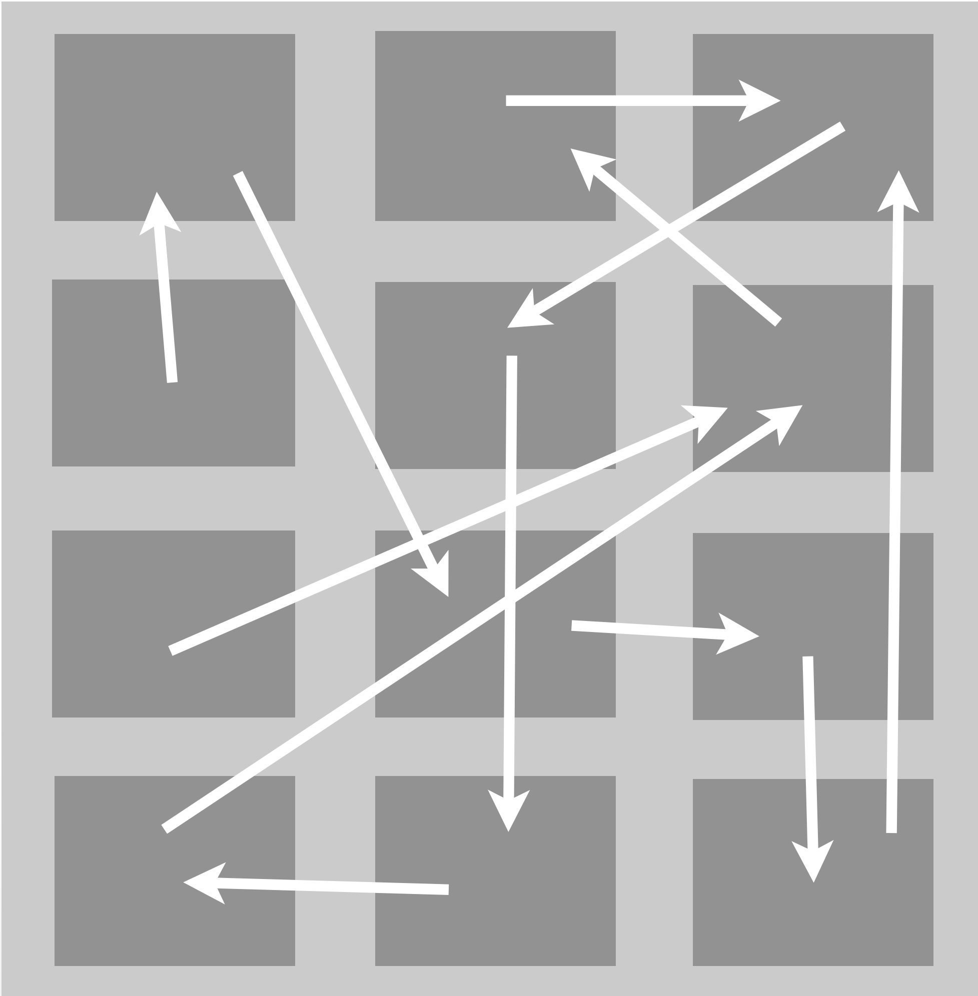
RANDOMIZED GOSSIP



Distributed Bulletin Board

- Each node keeps vector with per-node info (own + info received from others)
- Once per time step, each node sends to 1 other randomly selected node a subset of its own vector entries (called “window”)
- Node merges received window entries into local vector (if newer)

RANDOMIZED GOSSIP



Distributed Bulletin Board

- Each node keeps vector with per-node info (own + info received from others)
- Once per time step, each node sends to 1 other randomly selected node a subset of its own vector entries (called "window")
- Node merges received window entries into local vector (if newer)

MOSIX: GOSSIP ALGORITHM

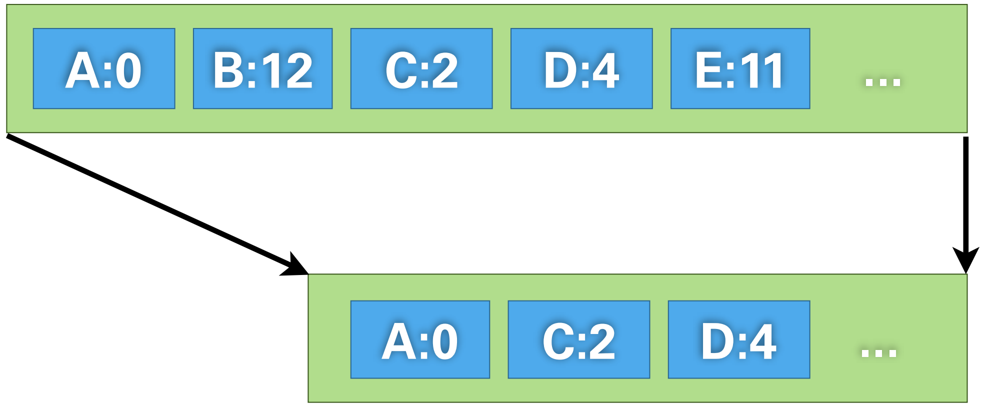
Each time unit:

- Update local info
- Find all vector entries up to age T (called a window)
- Send window to 1 randomly selected node

MOSIX: GOSSIP ALGORITHM

Each time unit:

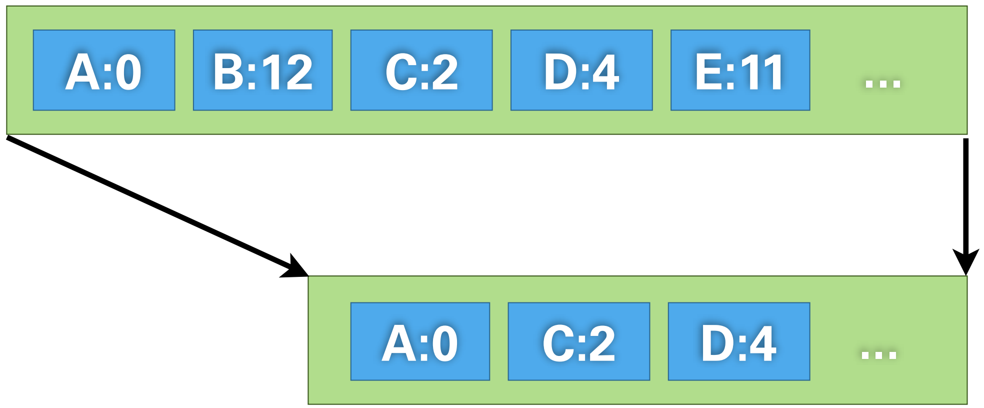
- Update local info
- Find all vector entries up to age T (called a window)
- Send window to 1 randomly selected node



MOSIX: GOSSIP ALGORITHM

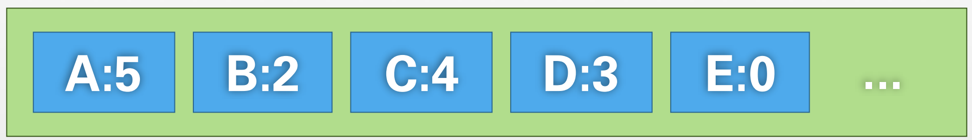
Each time unit:

- Update local info
- Find all vector entries up to age T (called a window)
- Send window to 1 randomly selected node



Upon receiving a window:

- Update the received entries' age (+1 for transfer)
- Update entries in local vector where newer information has been received



MOSIX: GOSSIP ALGORITHM

Each time unit:

- Update local info
- Find all vector entries up to age T (called a window)
- Send window to 1 randomly selected node

A:0	B:12	C:2	D:4	E:11	...

Upon receiving a window:

- Update the received entries' age (+1 for transfer)
- Update entries in local vector where newer information has been received

A:0	C:2	D:4	...

A:5	B:2	C:4	D:3	E:0	...

MOSIX: GOSSIP ALGORITHM

Each time unit:

- Update local info
- Find all vector entries up to age T (called a window)
- Send window to 1 randomly selected node

A:0	B:12	C:2	D:4	E:11	...

Upon receiving a window:

- Update the received entries' age (+1 for transfer)
- Update entries in local vector where newer information has been received

A:1	C:3	D:5	...

A:5	B:2	C:4	D:3	E:0	...

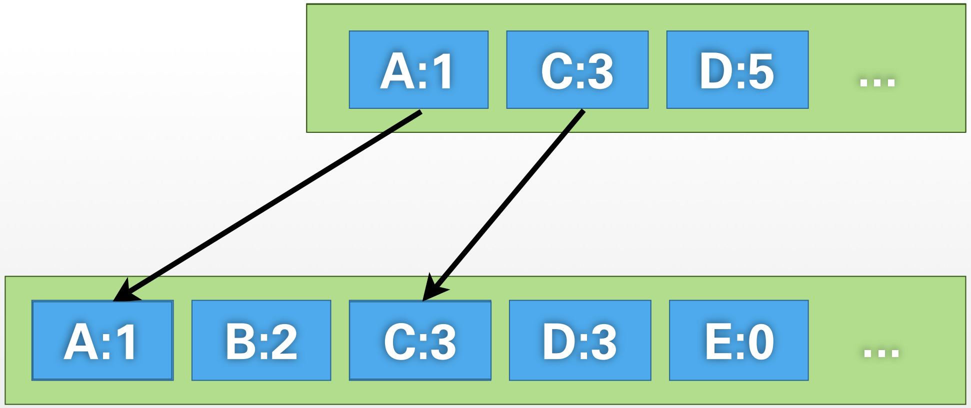
MOSIX: GOSSIP ALGORITHM

Each time unit:

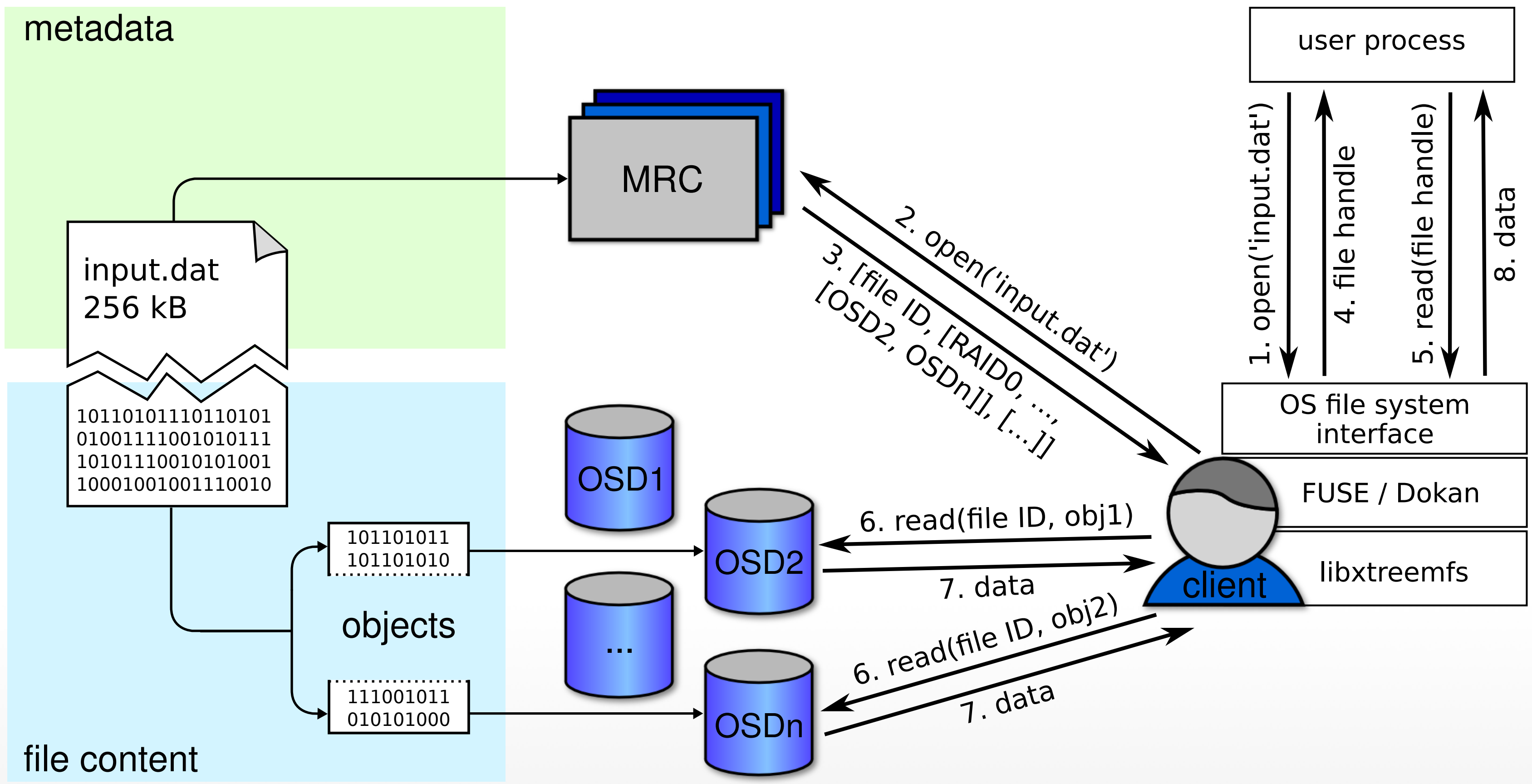
- Update local info
- Find all vector entries up to age T (called a window)
- Send window to 1 randomly selected node

Upon receiving a window:

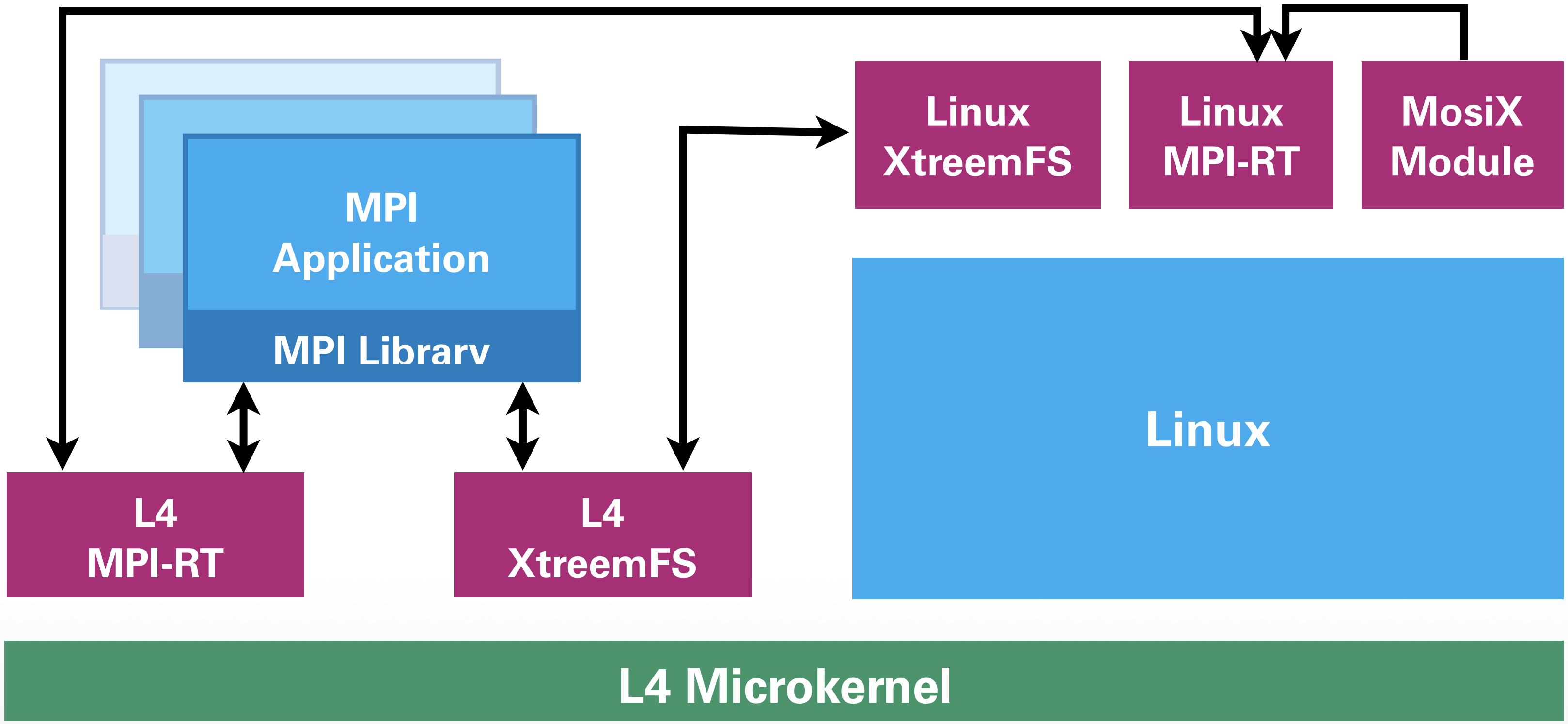
- Update the received entries' age (+1 for transfer)
- Update entries in local vector where newer information has been received



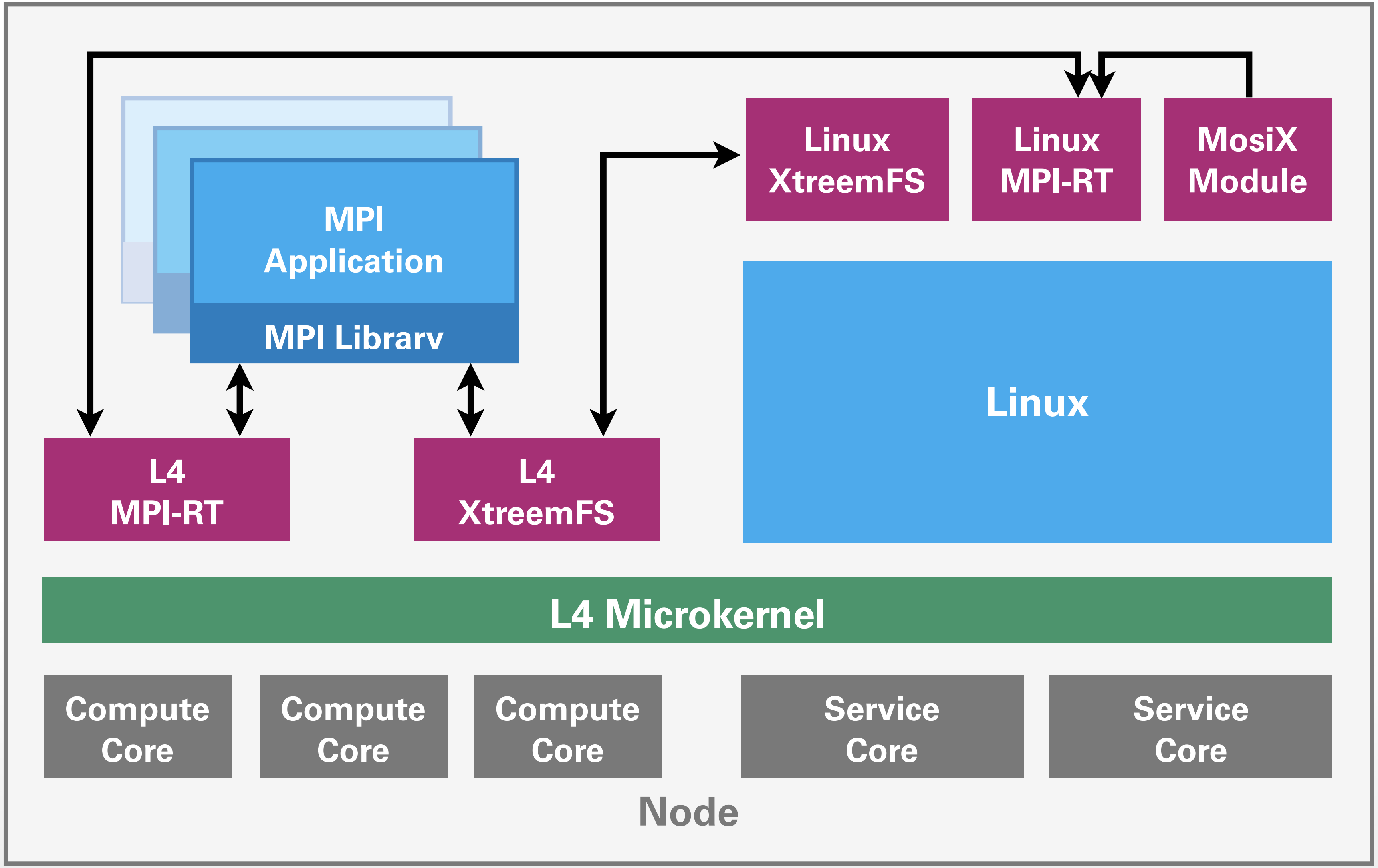
XTREEMFS ARCHITECTURE



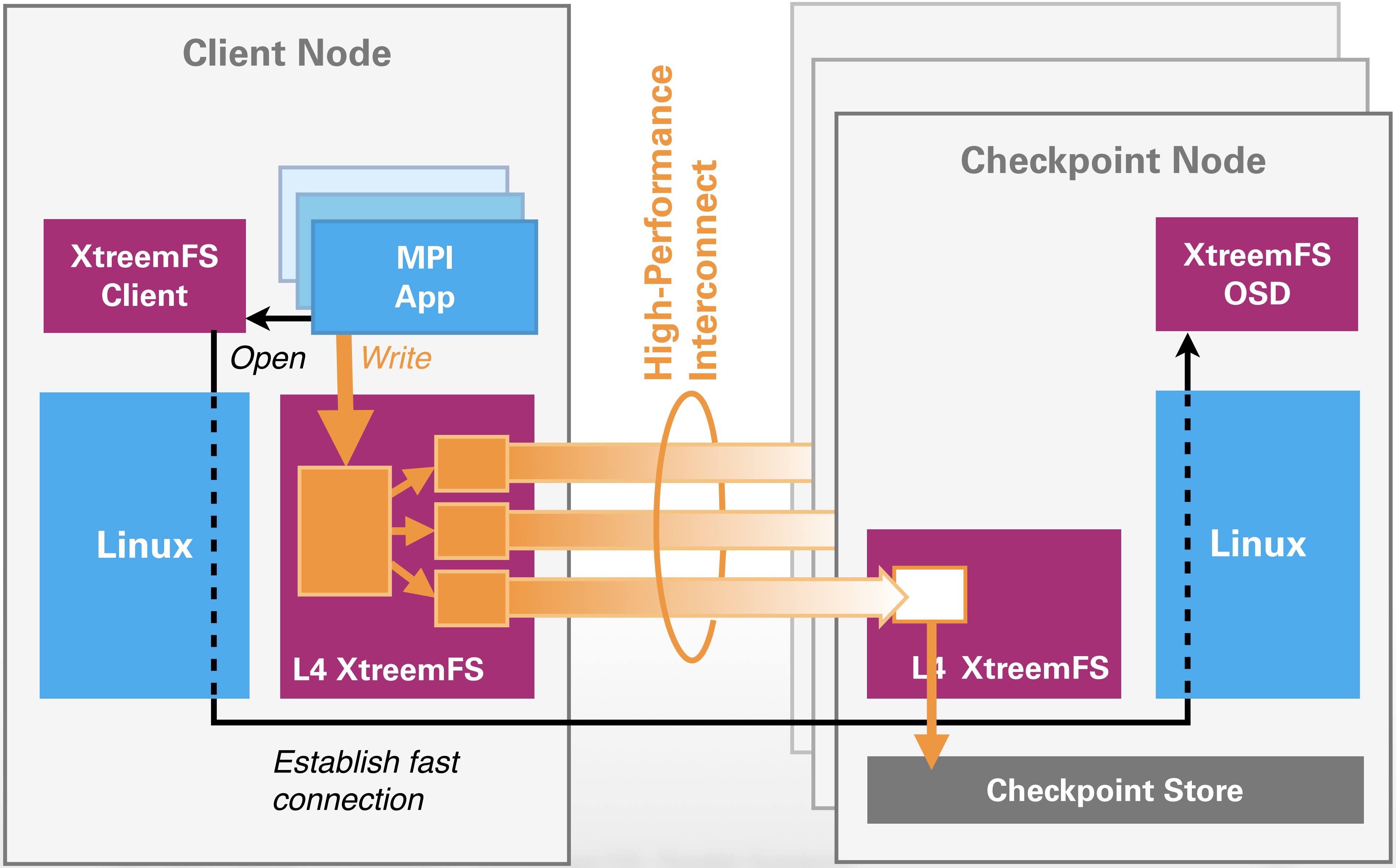
ARCHITECTURE



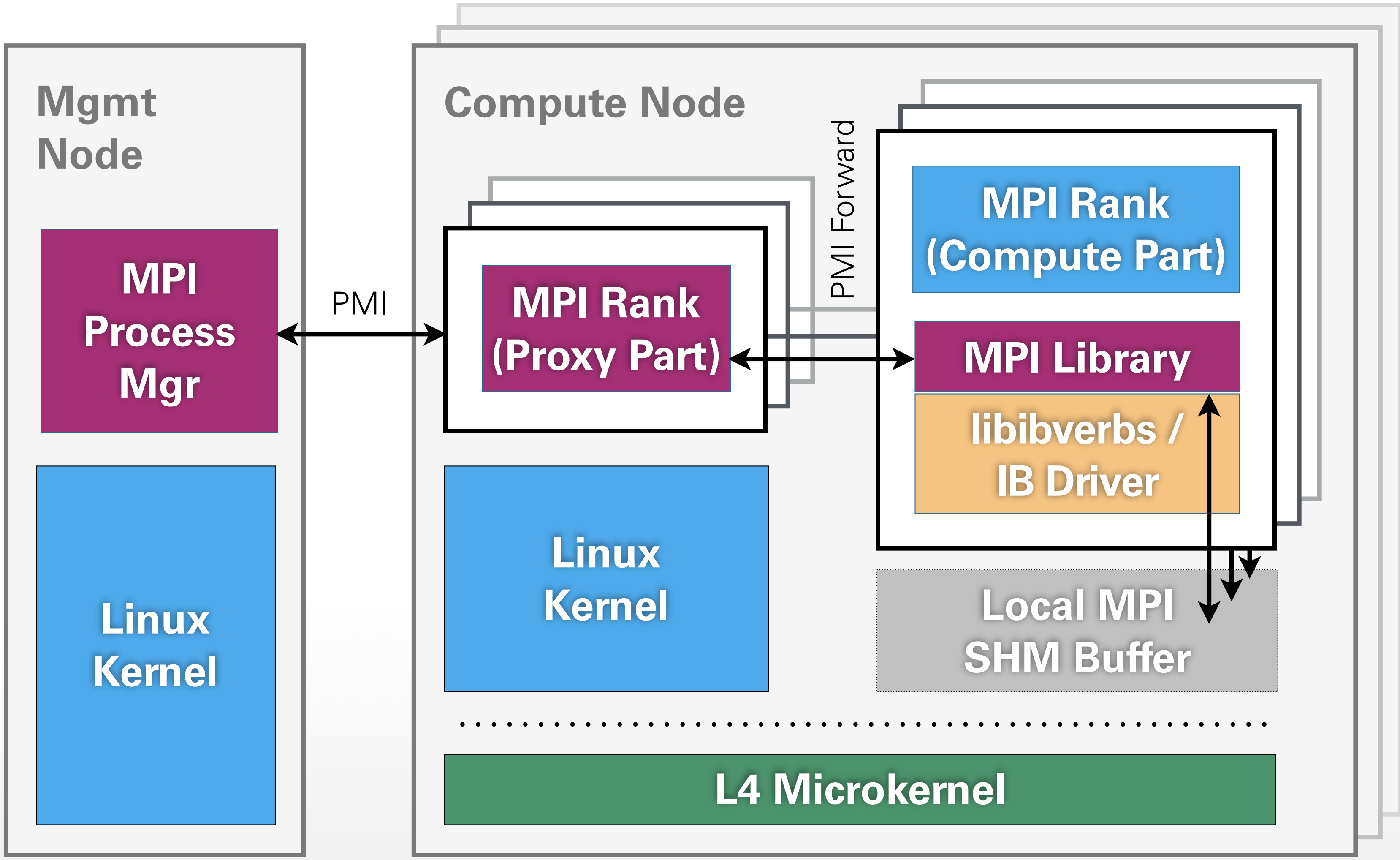
ARCHITECTURE



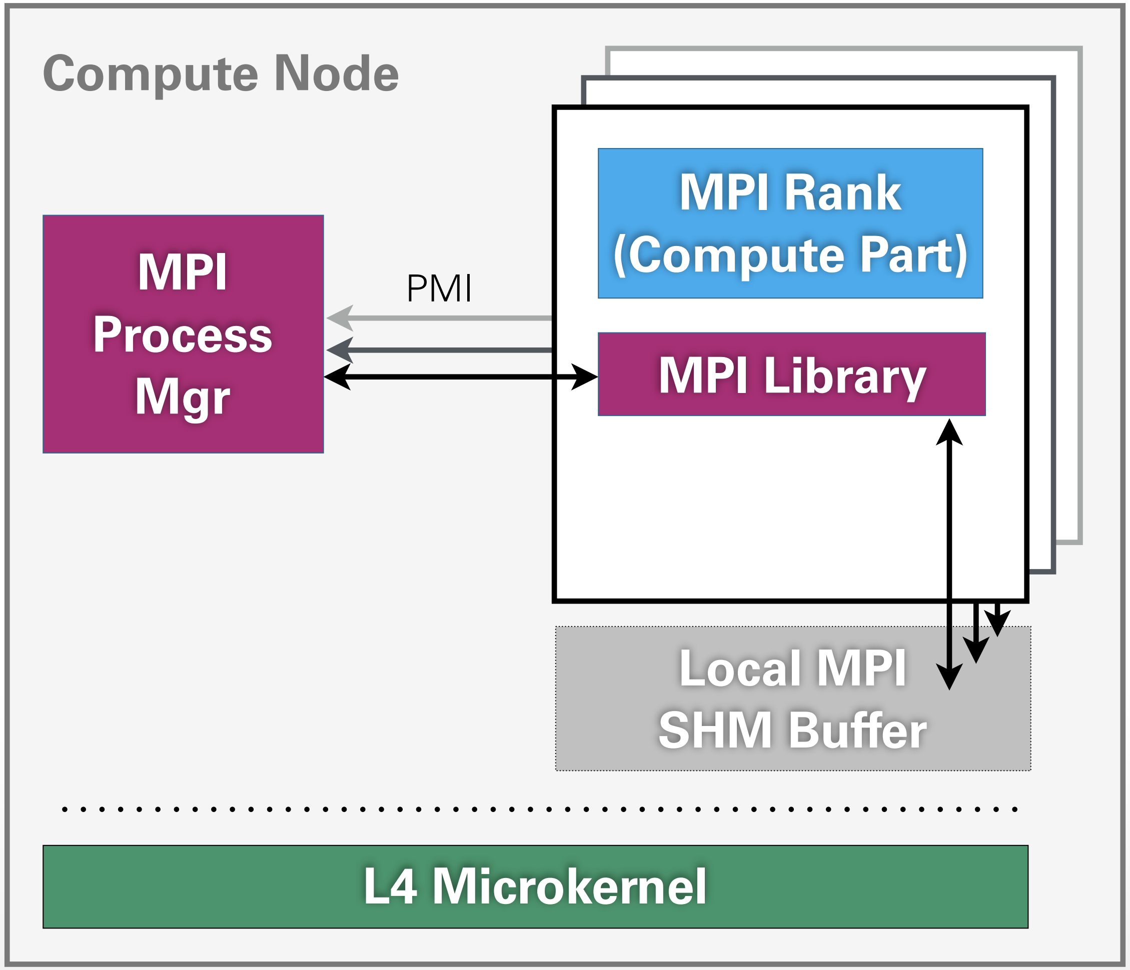
XTREEMFS: FAST PATH



SPLIT MPI ARCHITECTURE



SPLIT MPI ARCHITECTURE



DESIGN CHALLENGES

CHALLENGES

Fine-grained work splitting for system-supported load balancing?

How to synchronize? RDMA + polling ./. Block?

Gossip + Heuristics for EXASCALE ?

Application / system interface? "Yell" for help?

Compute processes, how and where to migrate / reroute communication?

Replication instead of / in addition to checkpoint/ restart?

Reuse Linux (device drivers)?

HARDWARE-WISHES

- Perf counters for Network
- fast redirection of messages
- flash on node circumventing FTL
- quick activation of threads without polling