TECHNISCHE UNIVERSITÄT DRESDEN

ZIB

The Hebrew University of Jerusalem

FFMK: BUILDING AN EXASCALE OPERATING SYSTEM

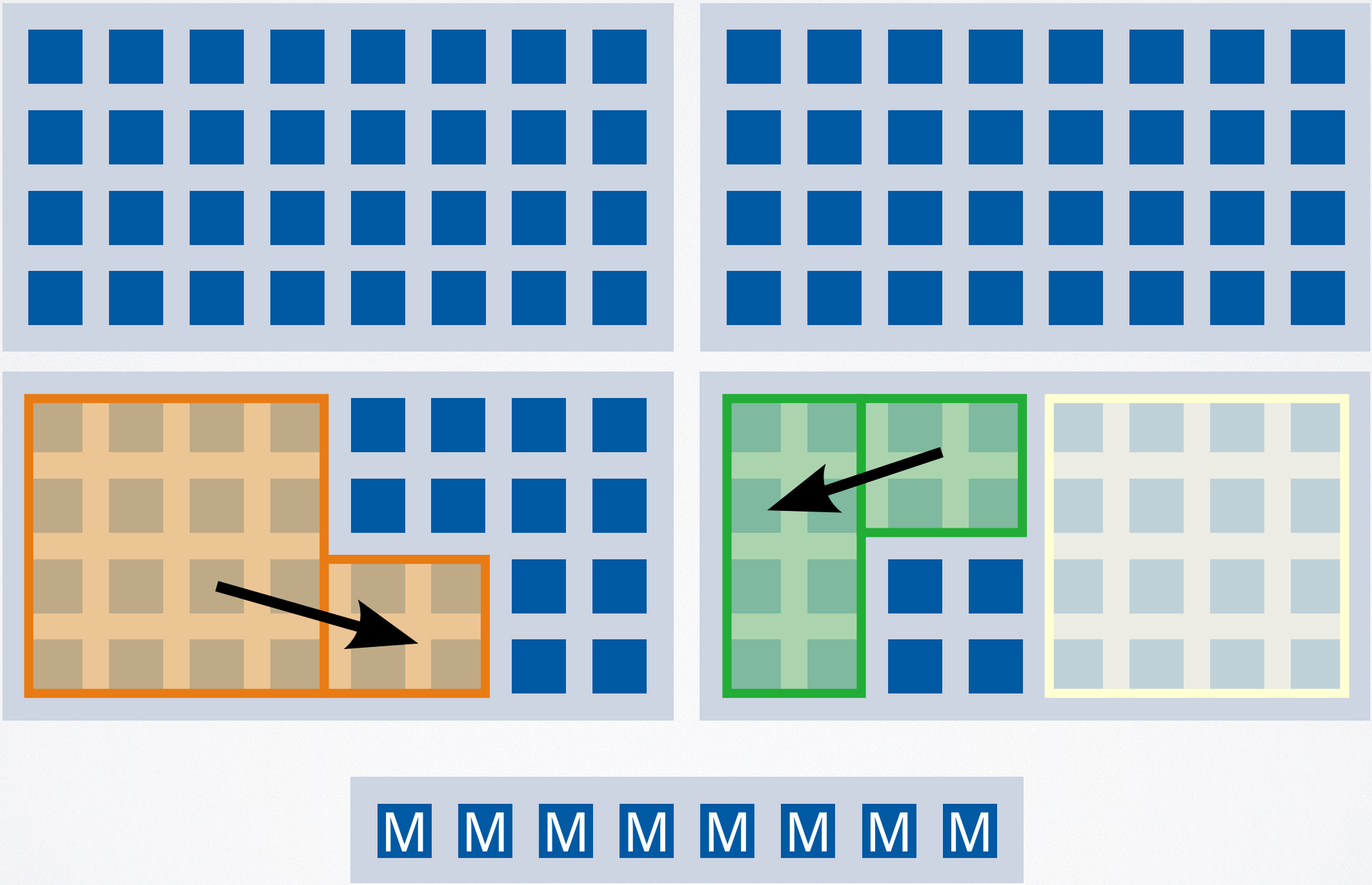
Amnon Barak	Hebrew University Jerusalem (HUJI)
Hermann Härtig	TU Dresden, Operating Systems Group (TUDOS)
Wolfgang E. Nagel	TU Dresden, Center for Information Services and HPC (ZIH)
Alexander Reinefeld	Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)

CARSTEN WEINHOLD, TU DRESDEN

CHALLENGES

- Heterogeneity (cores, memory, …)
- Power / dark silicon
- Increased failure rates
- „OS Noise“
- Dynamic platforms & applications

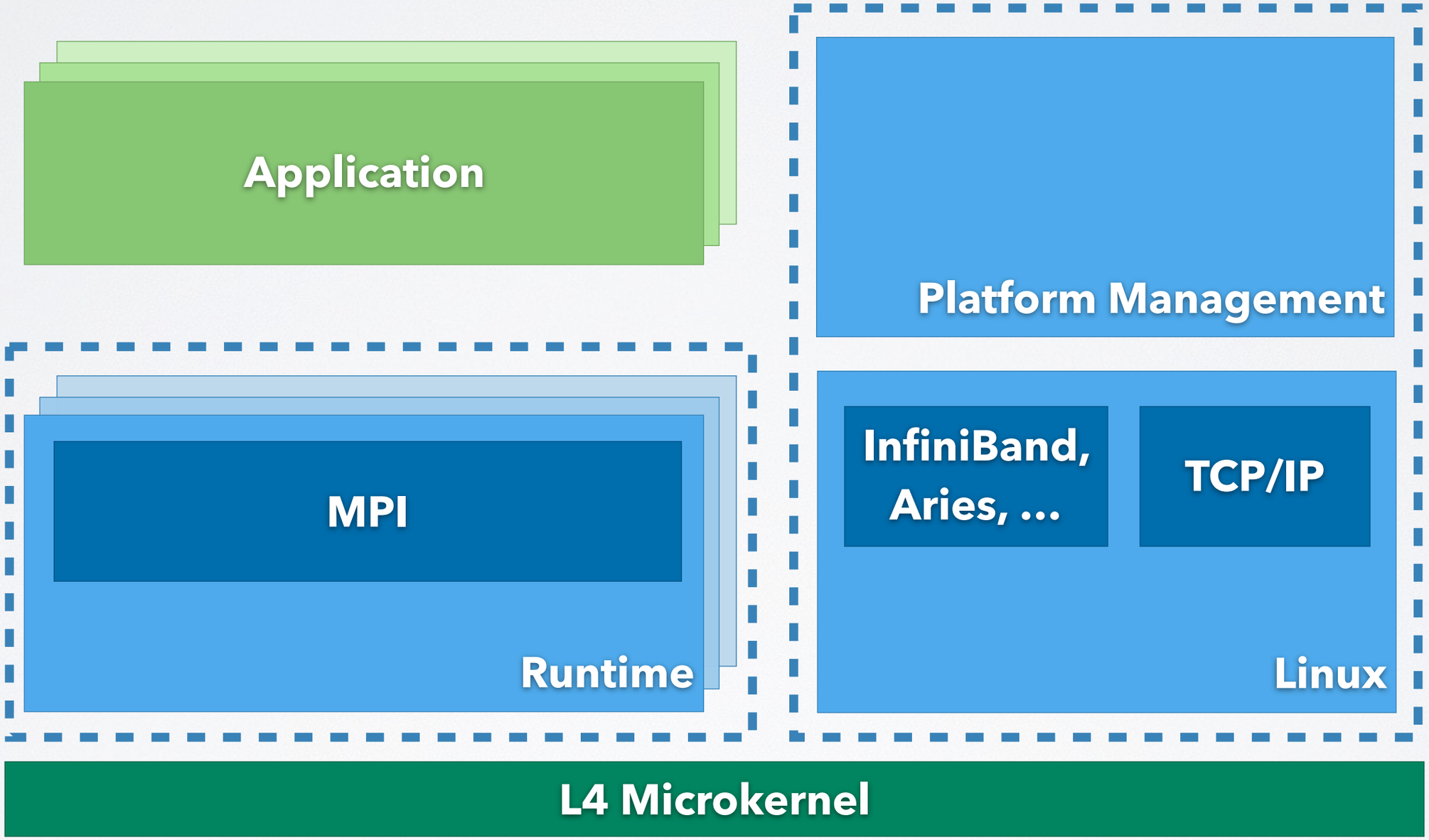
SYSTEM MODEL



OUTLINE

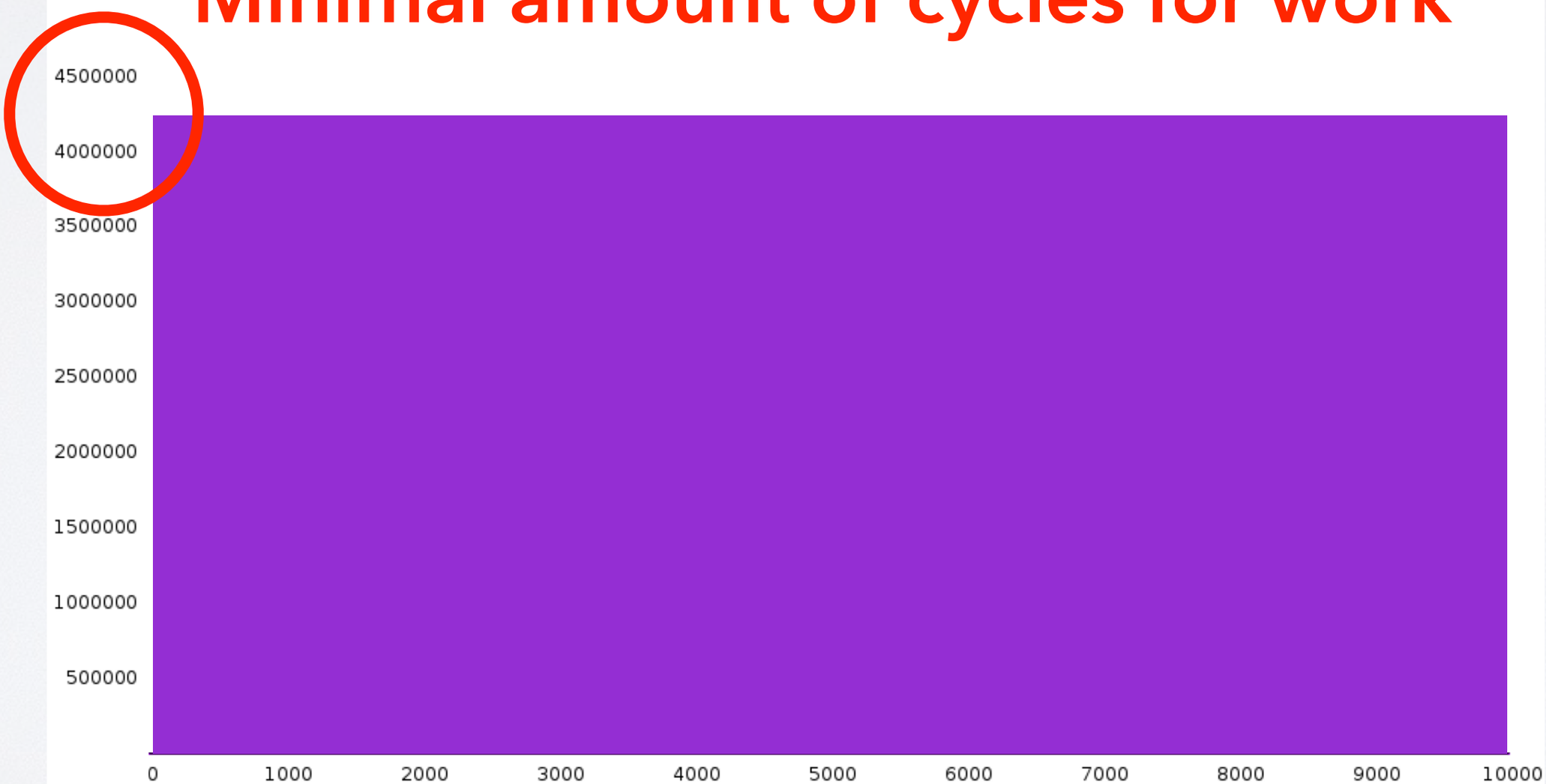
- **Architecture**
- **Three main results:**
  - Runs on a real system
  - Execution model + noise
  - Oversubscription / load balancing
- **Outlook:** Application hints
- Summary

NODE ARCHITECTURE

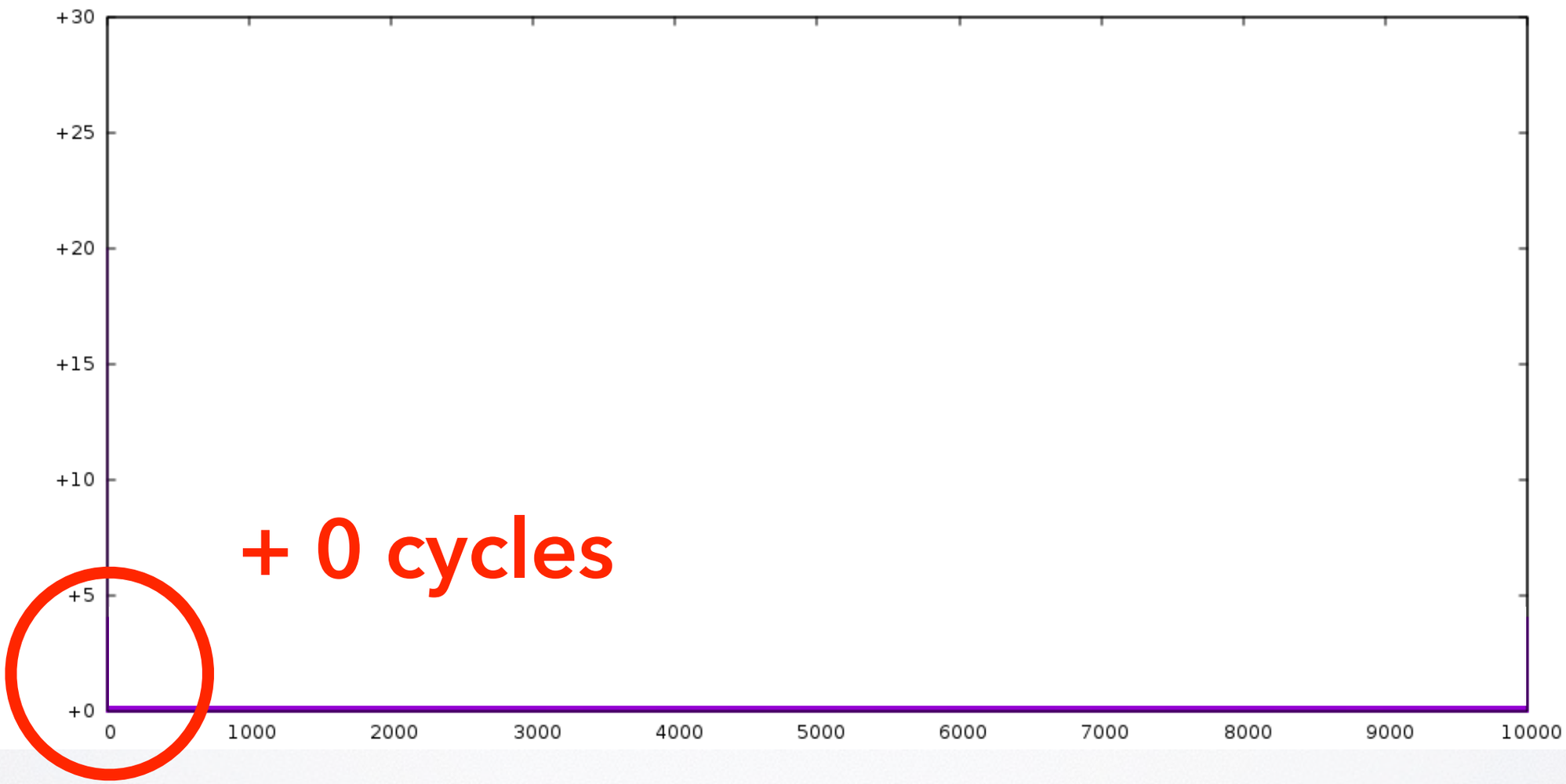


FWQ BENCHMARK

Minimal amount of cycles for work

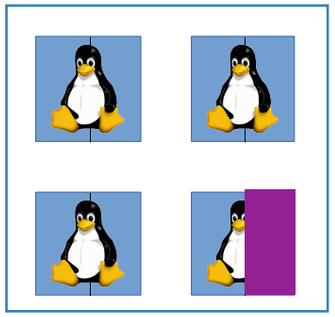


Ideal: zero extra cycles

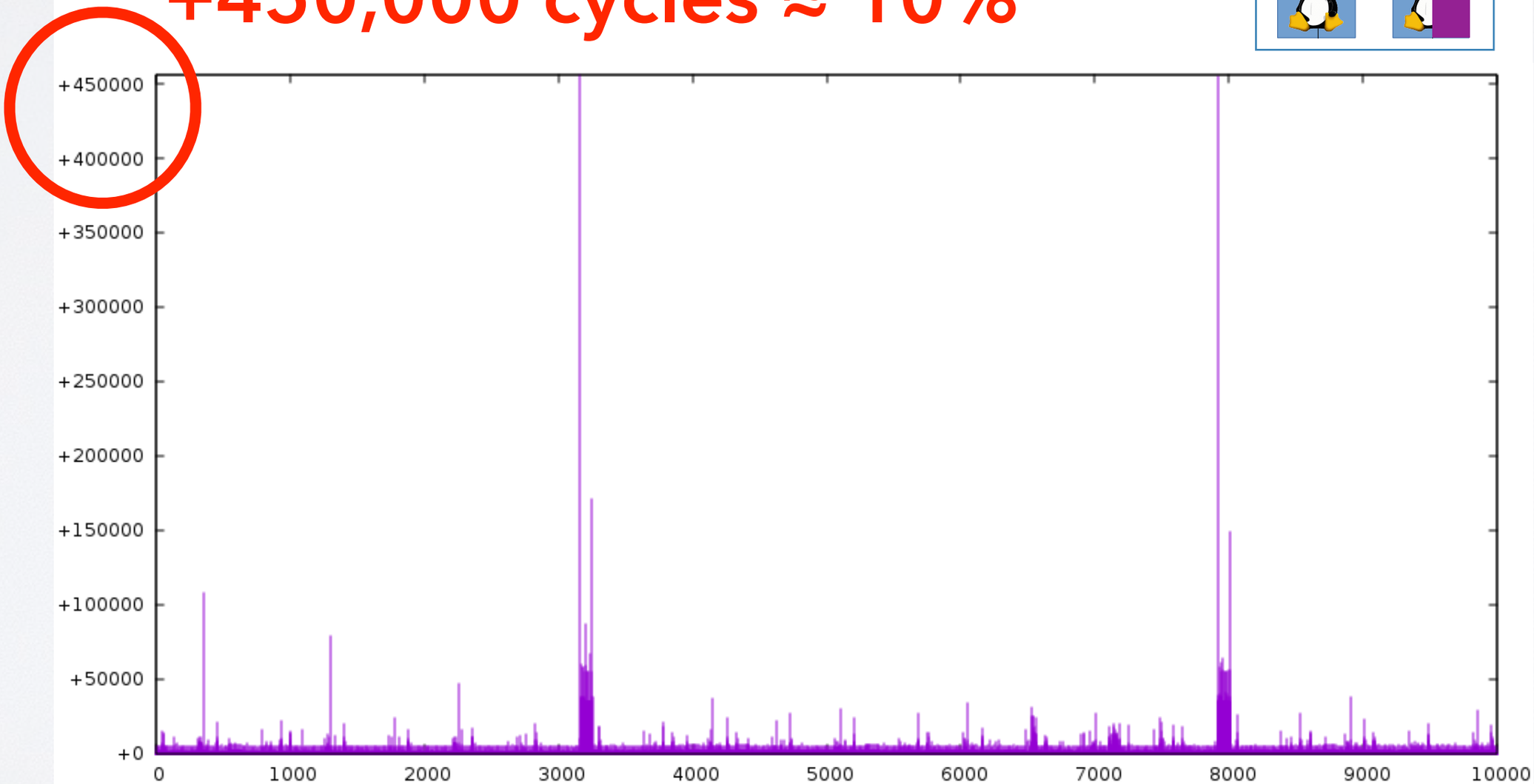


NOISY LINUX?

Real-World HPC Linux

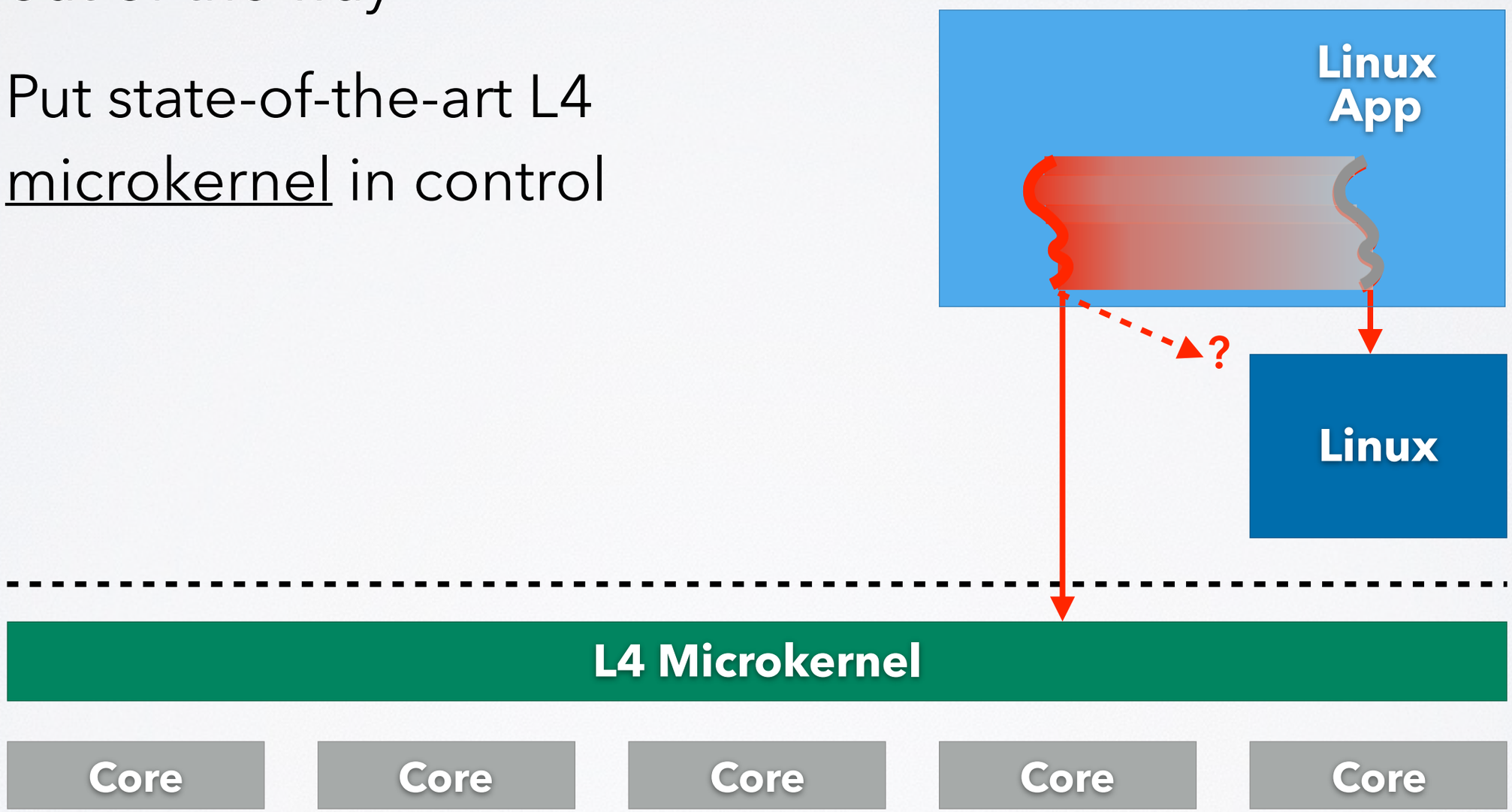


+450,000 cycles ≈ 10%

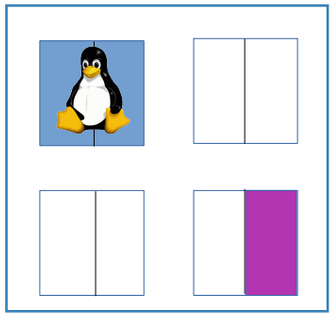


EXECUTION MODEL

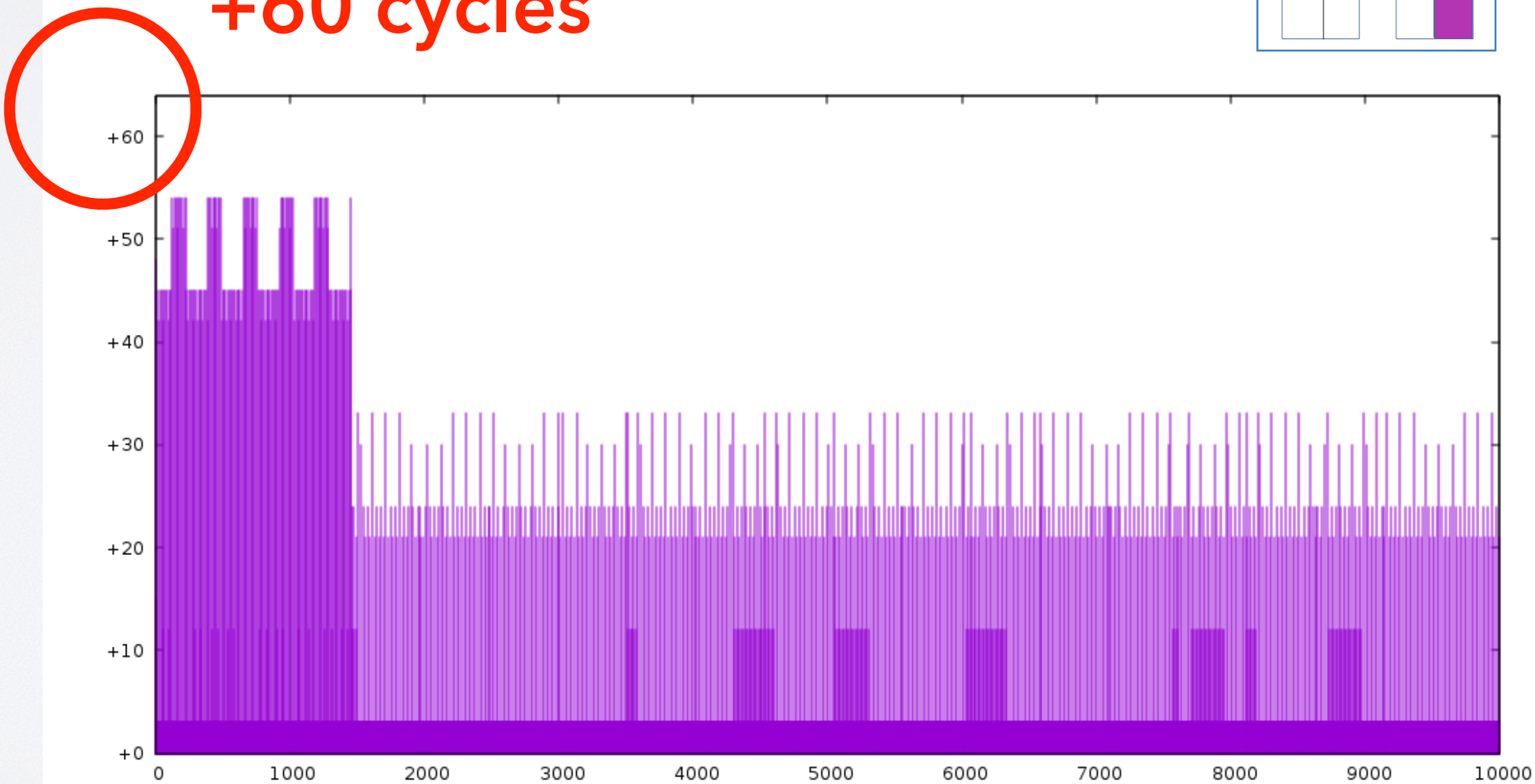
- Use Linux, but move it out of the way
- Put state-of-the-art L4 microkernel in control



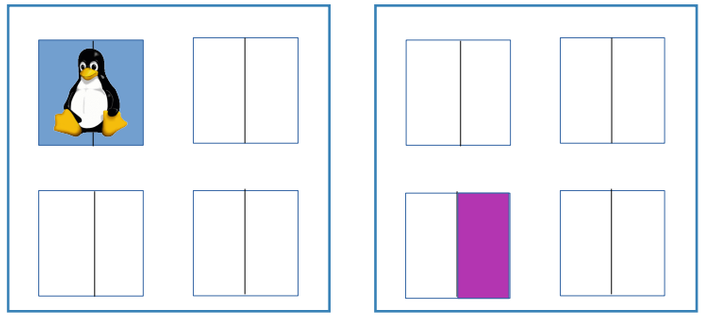
Decoupled Linux thread

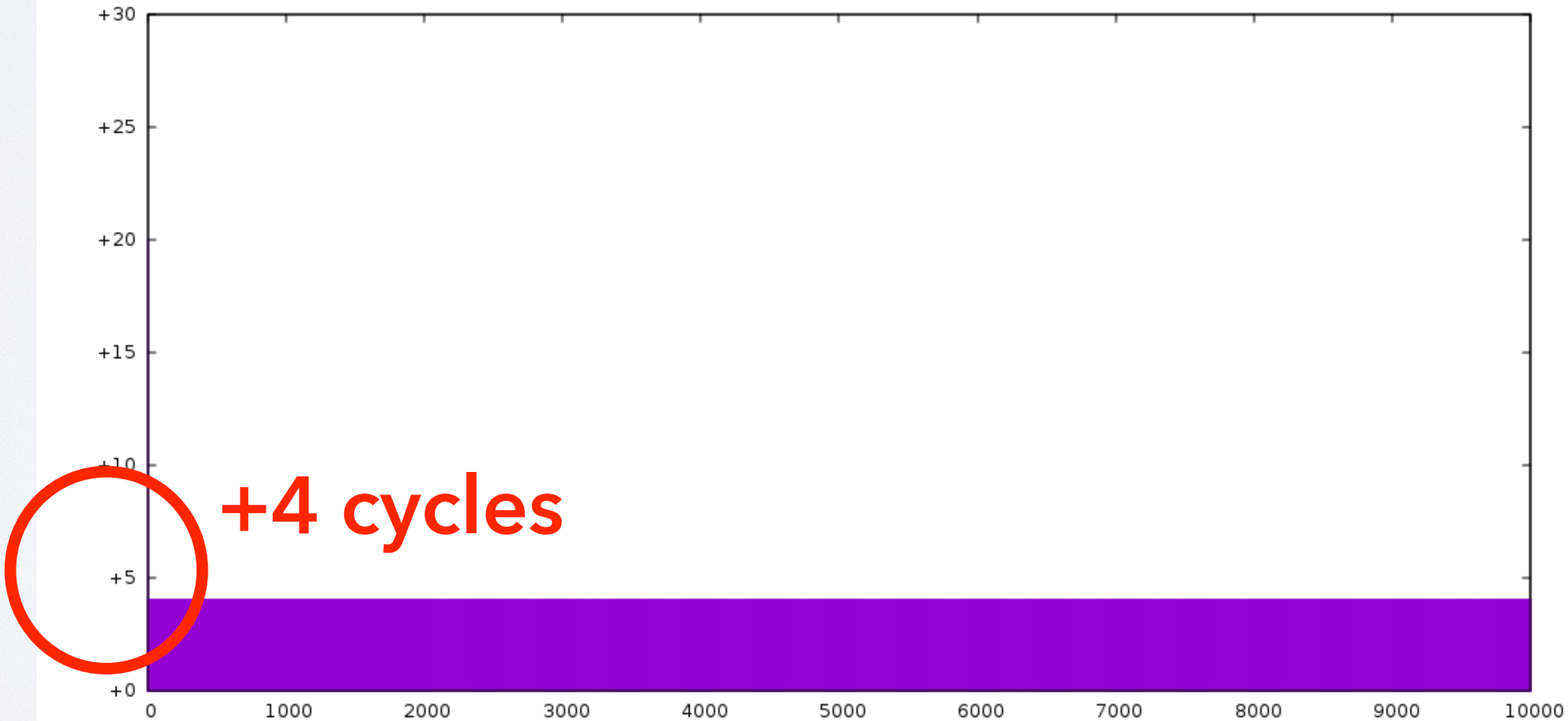


+60 cycles



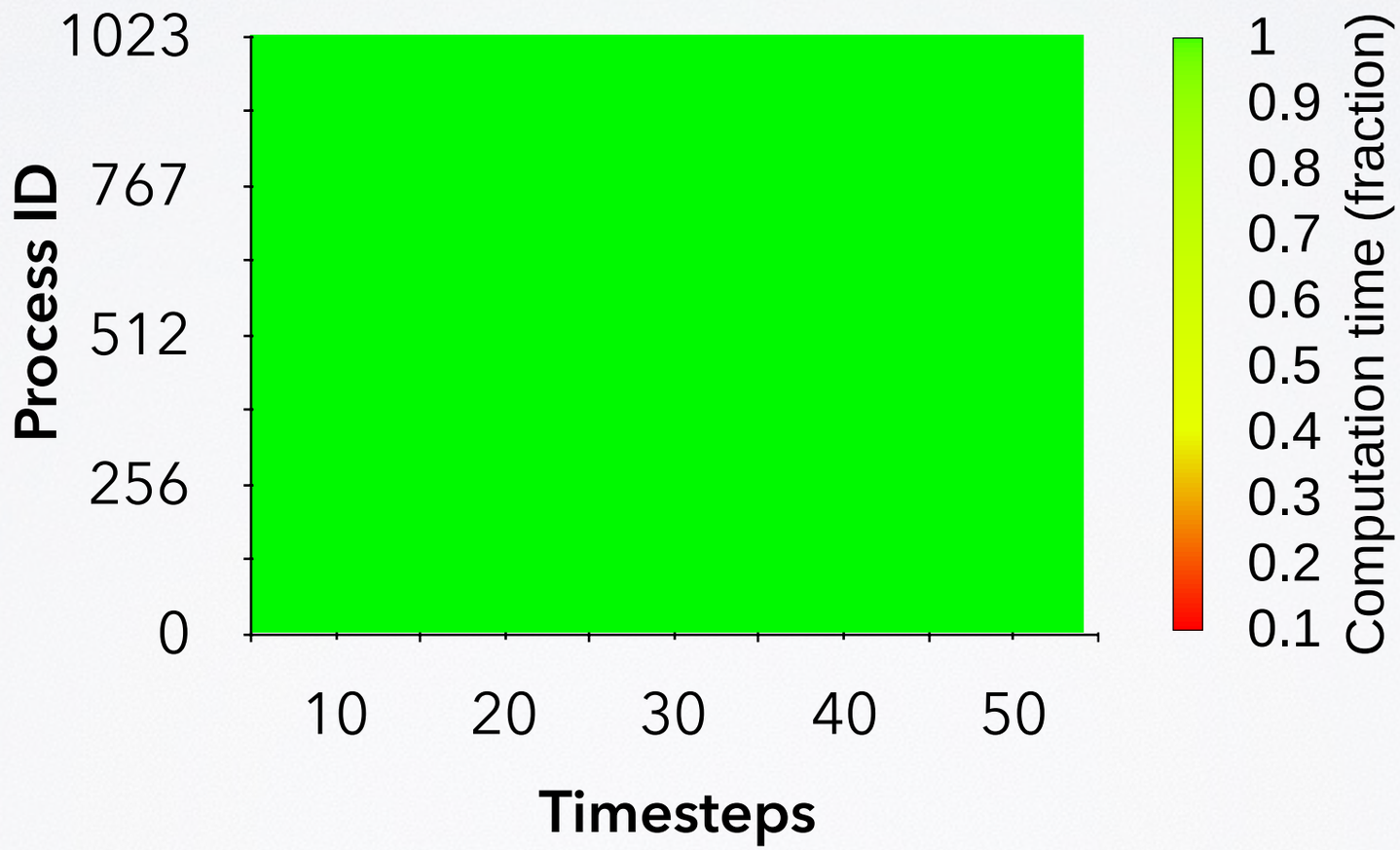
Decoupled Linux thread





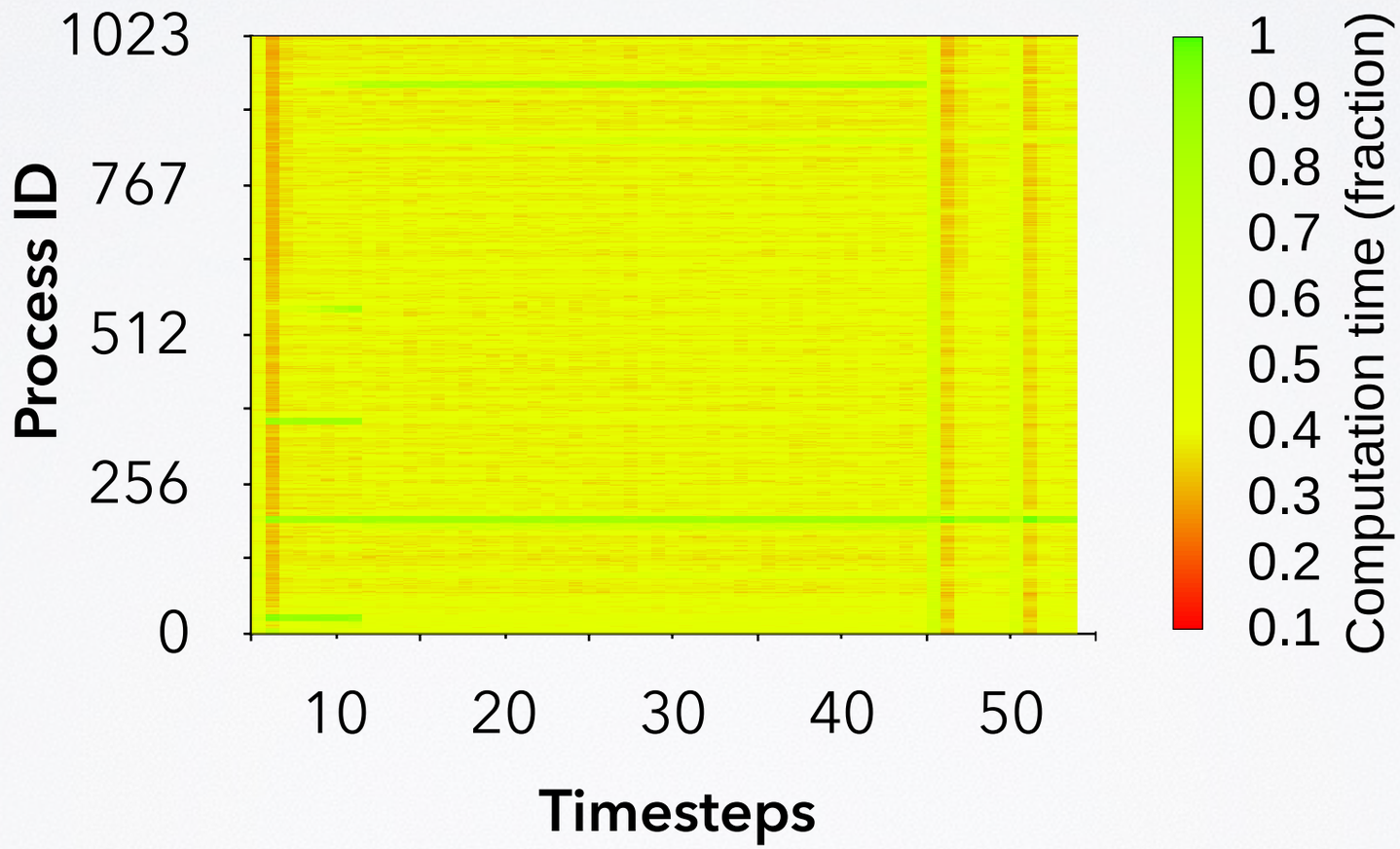
LOAD IMBALANCES

Ideal behavior



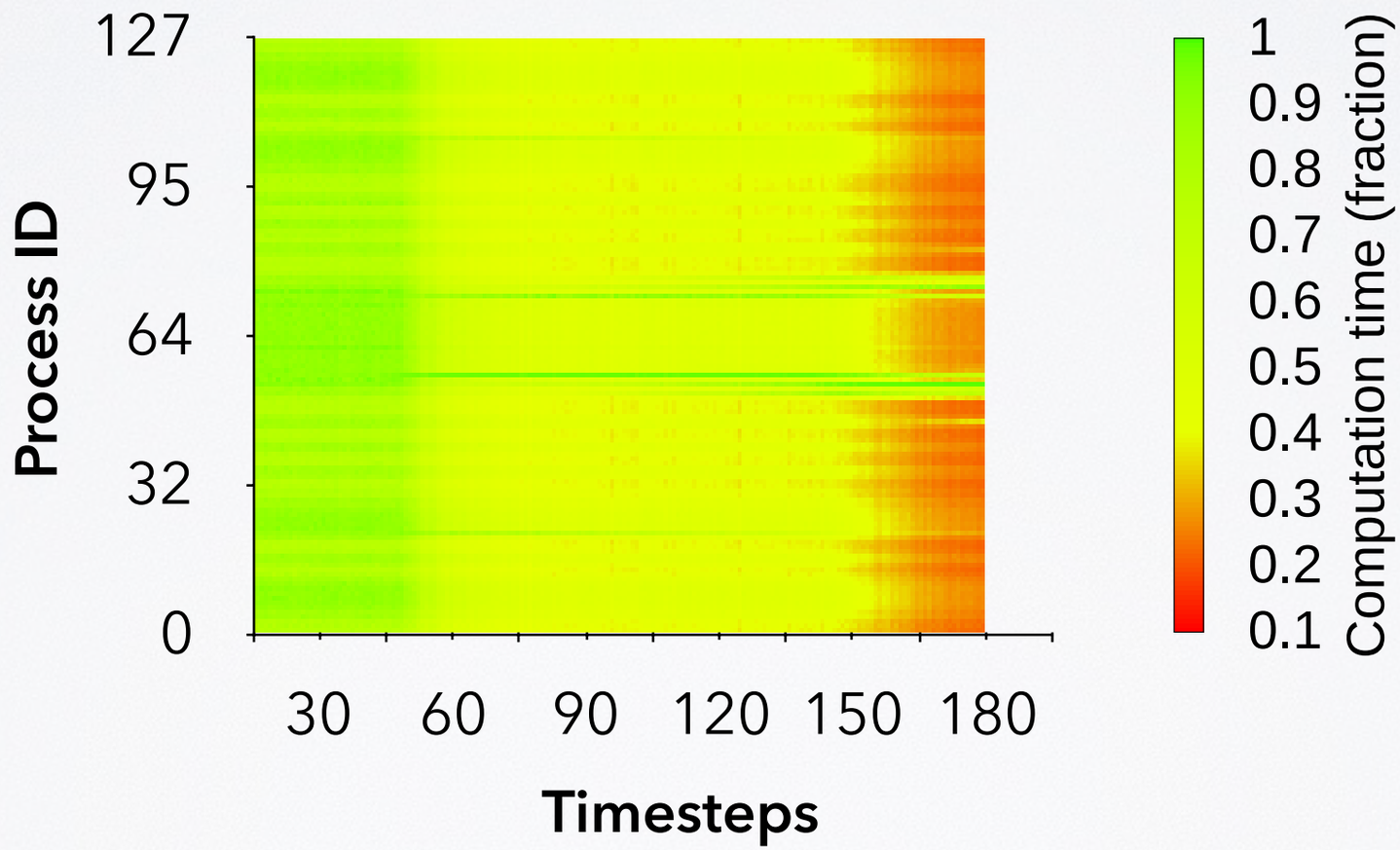
LOAD IMBALANCES

Application: CP2K

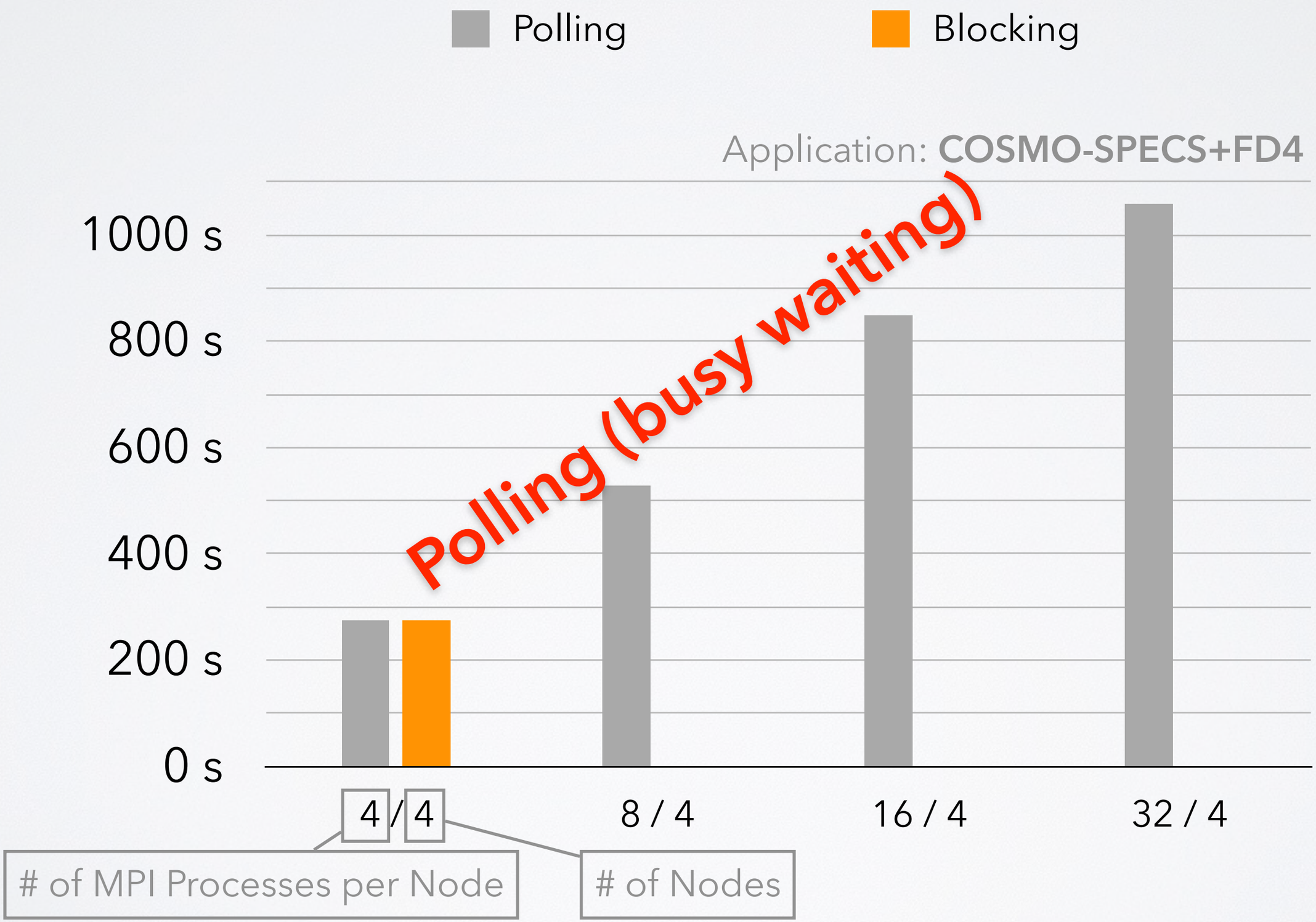


LOAD IMBALANCES

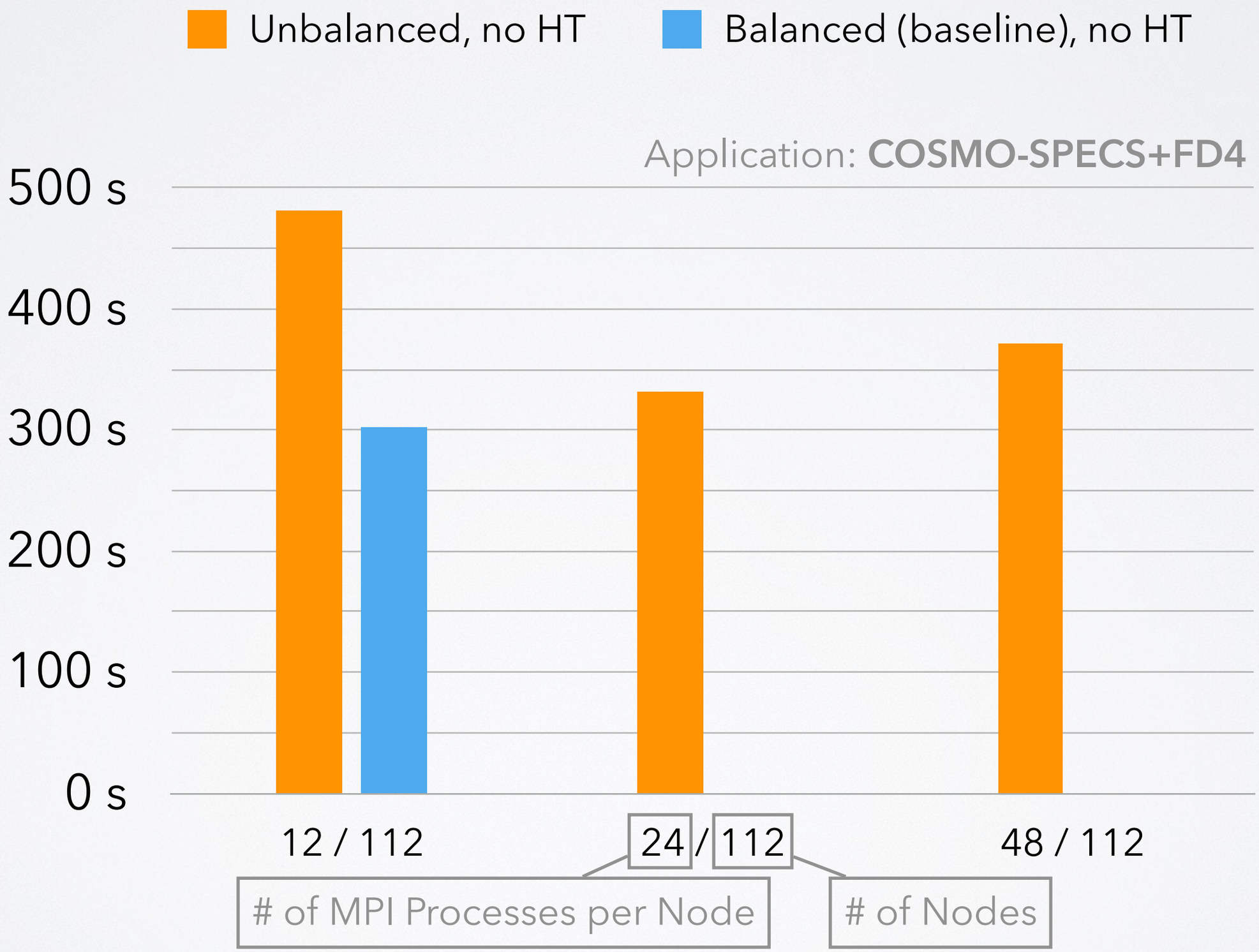
Application: COSMO-SPECS+FD4



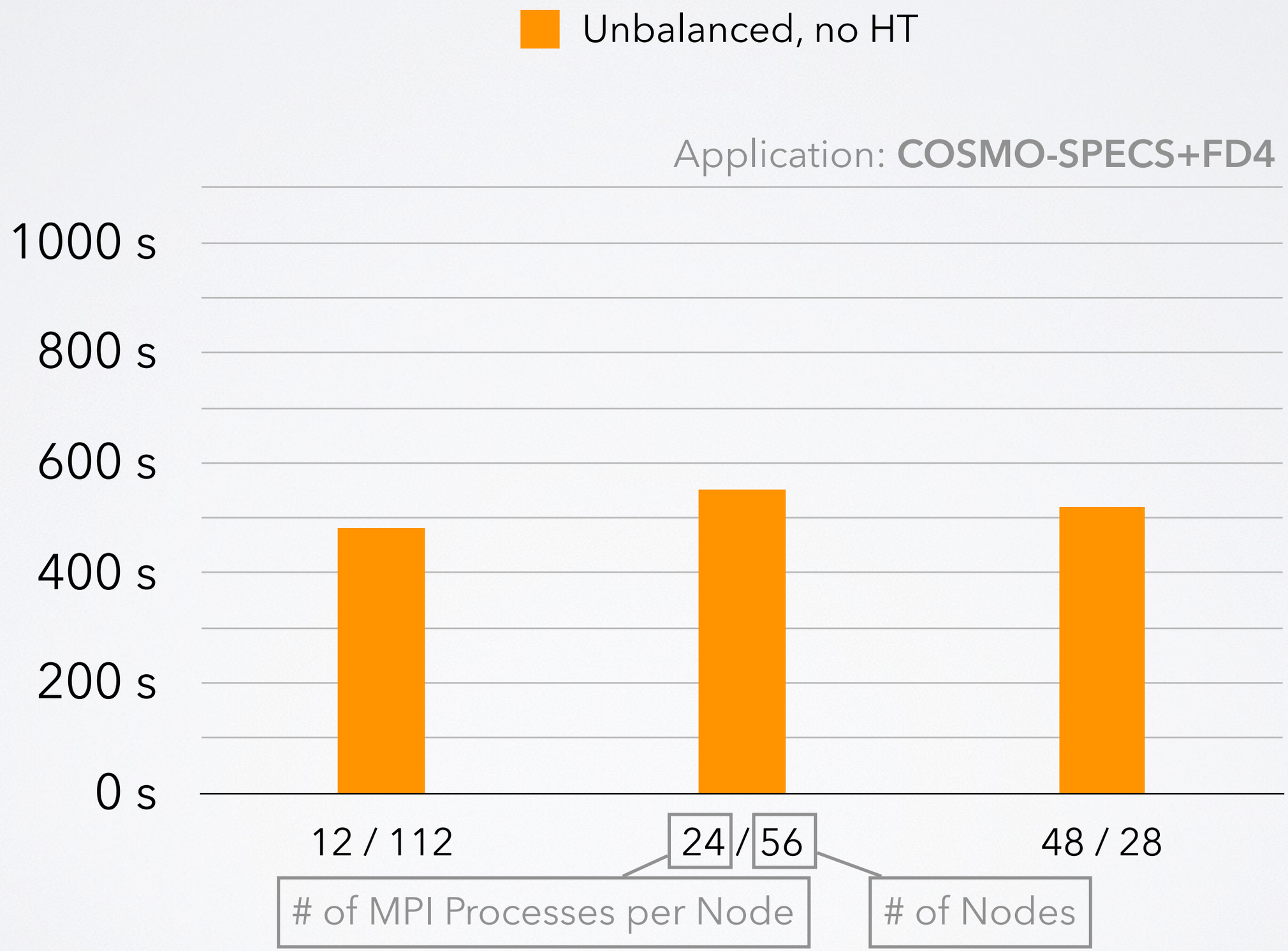
OVERDECOMPOSITION



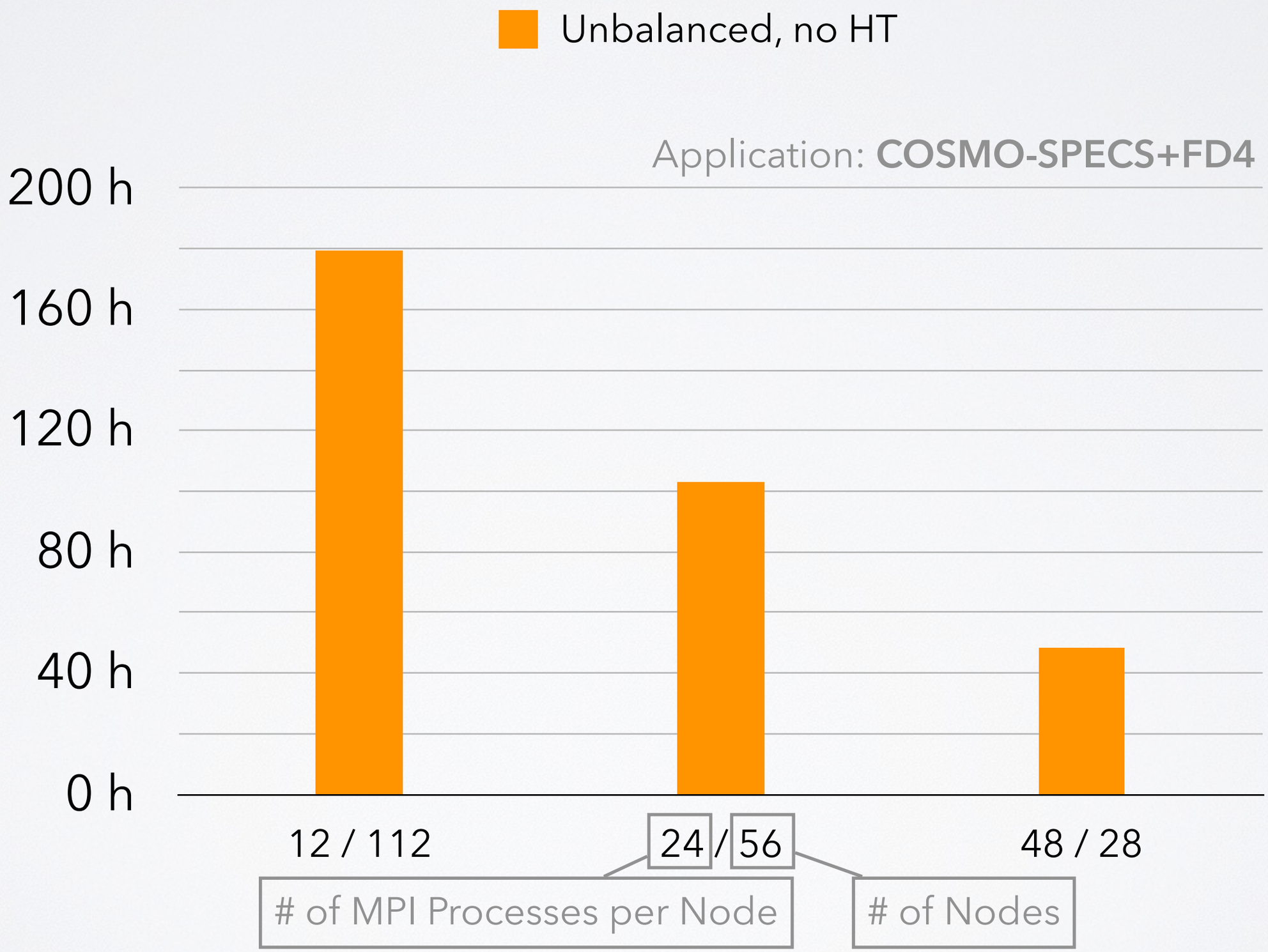
RUN TIME


Unbalanced, no HT
Balanced (baseline), no HT
Application: COSMO-SPECS+FD4
500 s
400 s
300 s
200 s
100 s
0 s
12 / 112
24 / 112
48 / 112
# of MPI Processes per Node
# of Nodes

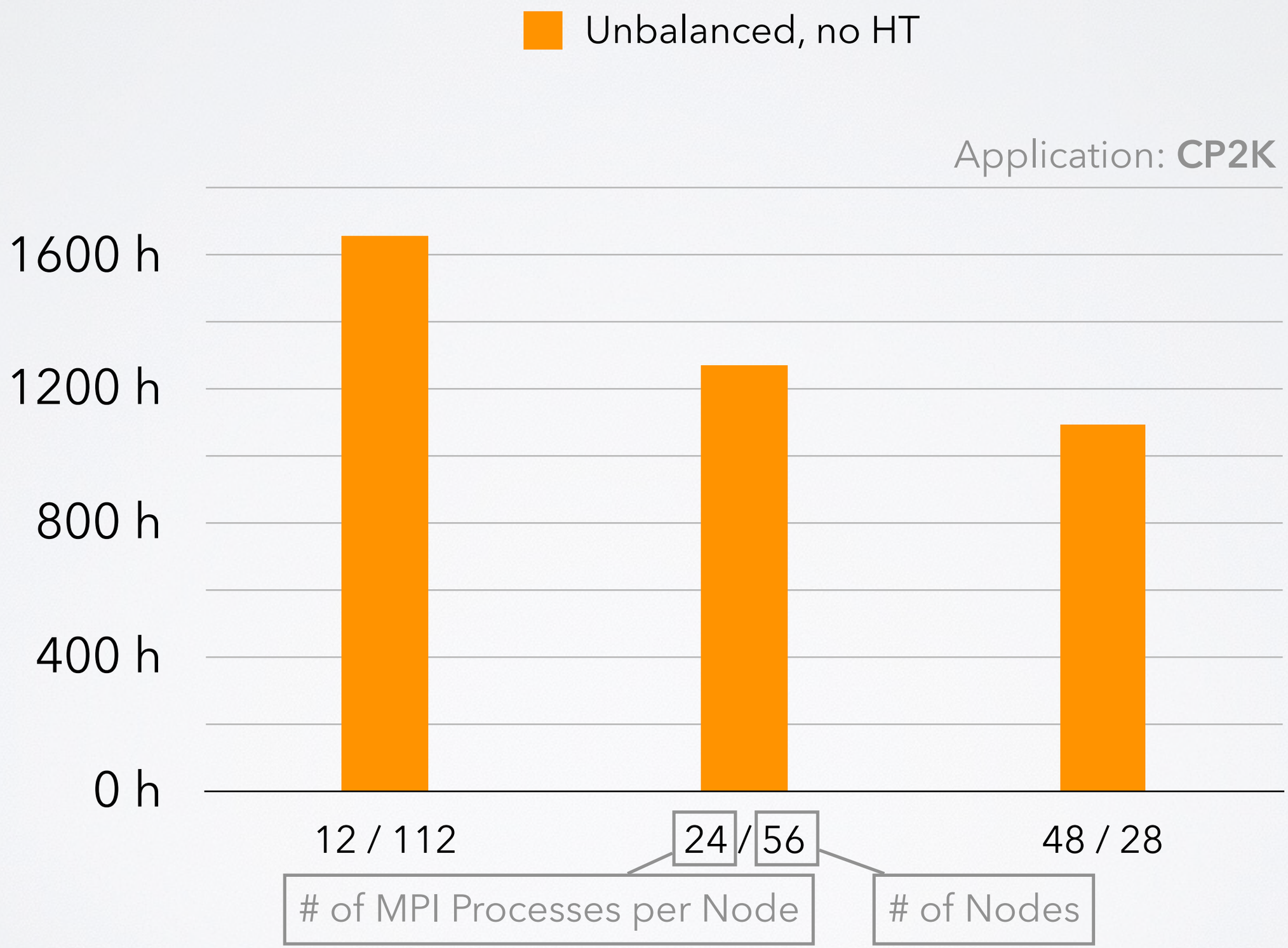
RUN TIME


Unbalanced, no HT
Application: COSMO-SPECS+FD4
1000 s
800 s
600 s
400 s
200 s
0 s
12 / 112
24 / 56
48 / 28
# of MPI Processes per Node
# of Nodes

CORE HOURS


Unbalanced, no HT
Application: COSMO-SPECS+FD4
200 h
160 h
120 h
80 h
40 h
0 h
12 / 112
24 / 56
48 / 28
# of MPI Processes per Node
# of Nodes

CORE HOURS


Unbalanced, no HT
Application: CP2K
1600 h
1200 h
800 h
400 h
0 h
12 / 112
24 / 56
48 / 28
# of MPI Processes per Node
# of Nodes

CONCLUSIONS

- Oversubscription does work
- Better utilization / throughput
- „Hyper Threading“ even better

- No migration yet …
- … but Gossip

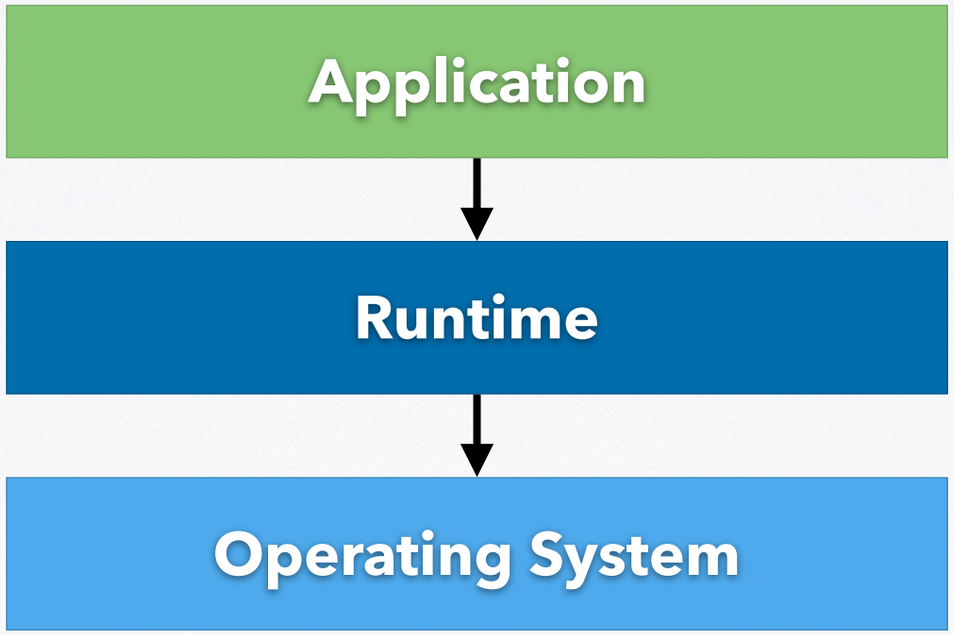
GOSSIP ALGORITHMS

- Spread load+health info among nodes
- Analytic model ~ simulation ~ emulation
- **Negligible overhead** (64–256 ms intervals)
- **Good quality of information** (2–3 s old)
- **Fault tolerant** (simulated for up to 32 of 1024 nodes failing)

E. Levy, A. Barak, A. Shiloh, M. Lieber, C. Weinhold, and H. Härtig, „Overhead of a Decentralized Gossip Algorithm on the Performance of HPC Applications“, ROSS 2014

A. Barak, Z. Drezner, E. Levy, M. Lieber, and A. Shiloh, „Resilient Gossip Algorithms for Collecting Online Management information in Exascale Clusters“, Concurrency and Computation: Practice and Experience, 2015

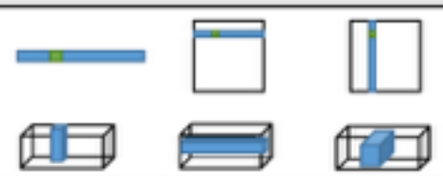
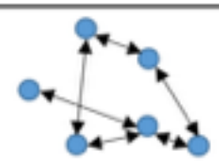
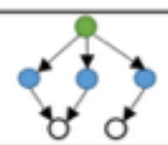
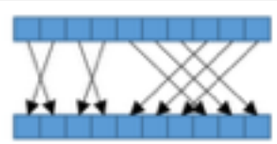
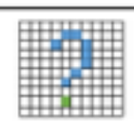
APPLICATION HINTS



APPLICATION HINTS

- Resource usage:
  - CPU cycles [energy?]
  - Cache misses
  - Memory [access patterns]

MAPS-MULTI

Access Pattern	Illustrations
Block (ND)	
Window (ND)	
Adjacency	
Traversal (BFS, DFS)	
Permutation	
Irregular	

- Template class for each access pattern
- Instantiate buffers (e.g., matrix) with appropriate class
- Framework / runtime optimizes data layout

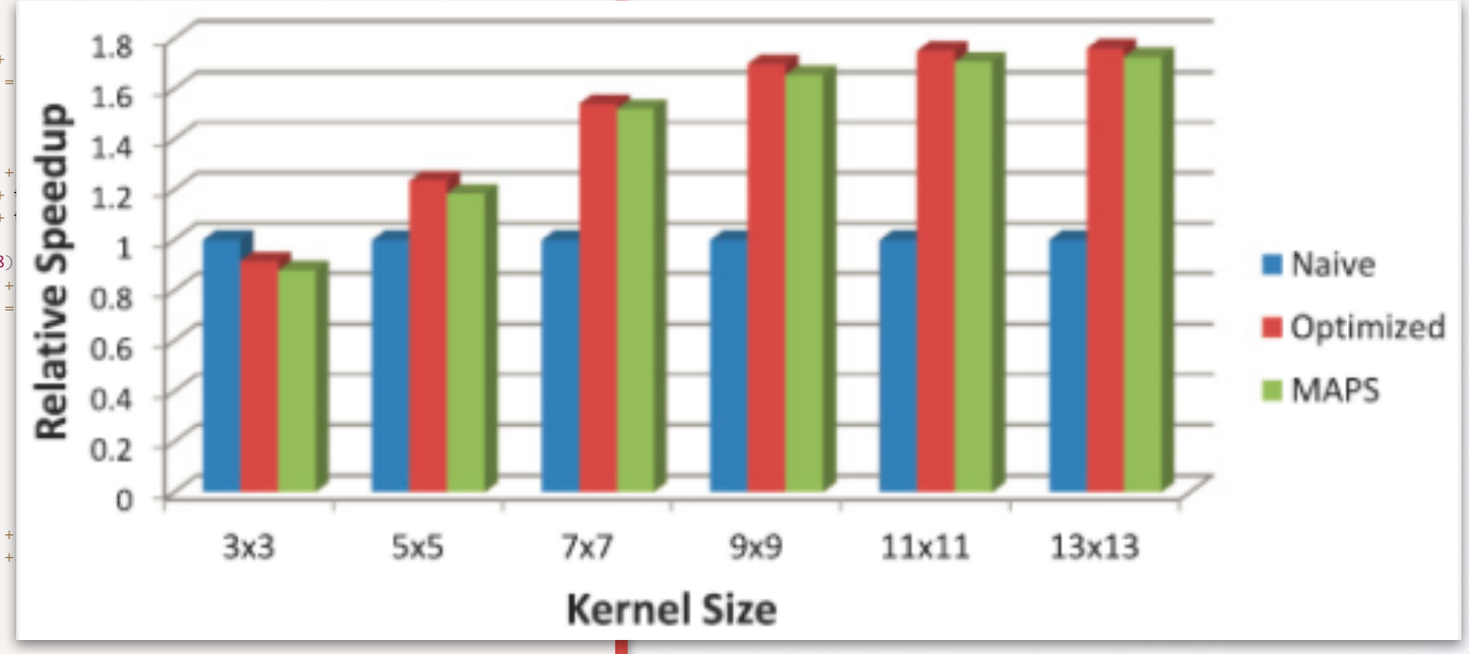
Tal Ben-Nun, Ely Levy, Amnon Barak, and Eri Rubin, „Memory Access Patterns: The Missing Piece of the Multi-GPU Puzzle", In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15, 2015

MAPS-MULTI

```
__global__ void Conv2(const float *in, int width, int height, int stride,
                      const float *convKernel, float *out) {
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    float result = 0.0f;
    for (int ky = -4; ky <= 4; ++ky) {
        for (int kx = -4; kx <= 4; ++kx) {
            result += in[CLAMP(y + ky, 0, height) * stride + CLAMP(x + kx, 0, width)] *
                      convKernel[(ky + 4) * 9 + (kx + 4)];
        }
    }

    out[y * stride + x] = result;
}
```

```
__constant__ float kConvKernel[9*9];

__global__ void
Conv2 MAPS_MULTIDEF(maps::Window2D<float, BLOCK_WIDTH, BLOCK_HEIGHT, RADIUS> in,
                    maps::StructuredInjective2D<float, BLOCK_WIDTH, BLOCK_HEIGHT> out) {
    MAPS_MULTI_INITVARS(in, out); // Initialize multi-GPU abstraction and containers

        // Loop over output elements
    #pragma unroll
    MAPS_FOREACH(oiter, out) {
       int i = 0;
        *oiter = 0.0f;

        // For each output, loop over inputs according to pattern
        #pragma unroll
        MAPS_FOREACH_ALIGNED(iter, in, oiter) {
           *oiter += *iter * kConvKernel[i++];
        }
    }

    out.commit(); // Write all outputs to global memory
}
```

```
__constant__ float kConvKernel[9*9];

__global__ void Conv2(const float *in, int width, int height, int stride,
                      const float *convKernel, float *out) {
    int x = blockIdx.x * BLOCK_WIDTH + threadIdx.x;
    int y = blockIdx.y * BLOCK_HEIGHT + threadIdx.y;

    __shared__ float s_temp[(BLOCK_WIDTH + 8) * (BLOCK_HEIGHT + 8)];

    s_temp[BLOCK_WIDTH * threadIdx.y + threadIdx.x] = in[CLAMP(y + threadIdx.y - 4, 0, height) * stride +

    if (threadIdx.x < 8) {
        s_temp[BLOCK_WIDTH * threadIdx.y +
               threadIdx.x + BLOCK_WIDTH] =

    }
    if (threadIdx.y < 8) {
        s_temp[BLOCK_WIDTH * (threadIdx.y +
               threadIdx.x] = in[CLAMP(y +
                                 CLAMP(x +
    }
    if (threadIdx.x < 8 && threadIdx.y < 8)
        s_temp[BLOCK_WIDTH * (threadIdx.y +
               threadIdx.x + BLOCK_WIDTH] =

    }
    __syncthreads();

    float result = 0.0f;
    #pragma unroll
    for (int ky = 0; ky < 9; ++ky) {
        #pragma unroll
        for (int kx = 0; kx < 9; ++kx) {
            result += s_temp[(BLOCK_WIDTH +
                              (threadIdx.x +
        }
    }

    out[y * stride + x] = result;
}
```



APPLICATION HINTS

- Resource prediction:
  - CPU cycles [energy?]
  - Cache misses
  - Memory [access patterns]
- Fault tolerance:
  - Which data to checkpoint (and when)
  - Ability to handle node failures

SUMMARY

- FFMK prototype runs on real HPC systems
- We build **OS/R for Exascale:**
  - Low noise
  - Load management
  - Fault tolerance
- Take burden from application developers

SPPEXA

ffmk.tudos.org