Barcelona Supercomputing Center
Centro Nacional de Supercomputación

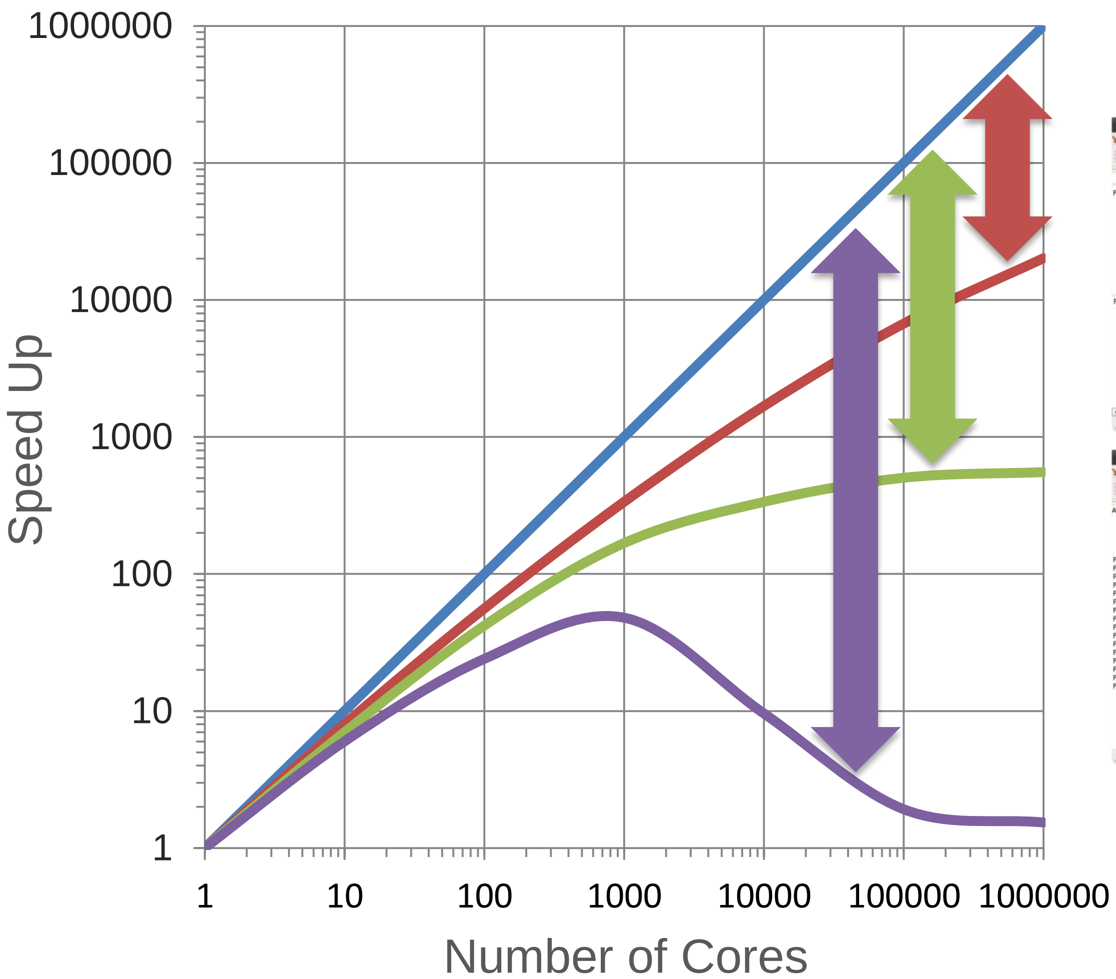
Automatic Adaption of the Sampling Frequency for Detailed Performance Analysis

Michael Wagner[1,2], Andreas Knüpfer[2]

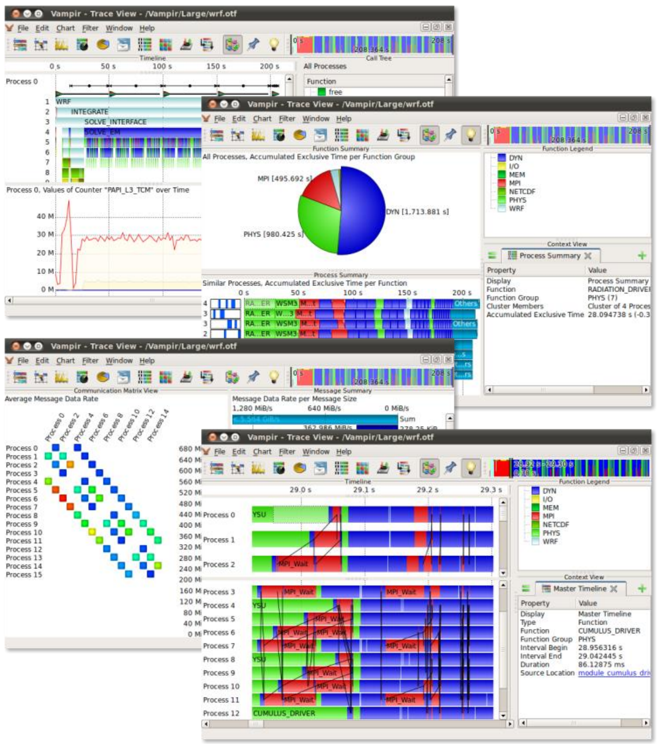
michael.wagner@bsc.es

1) Barcelona Supercomputing Center (BSC), Barcelona, Spain
2) Center for Information Services and High Performance Computing (ZIH), Dresden, Germany

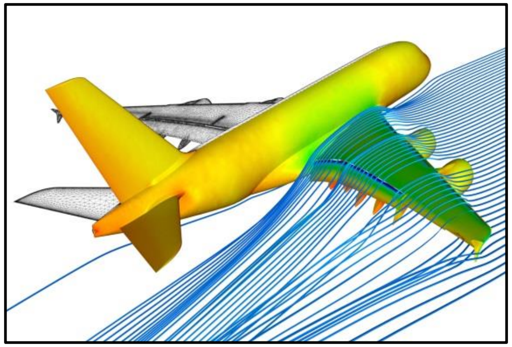
Parallelization – Ideal vs. Reality

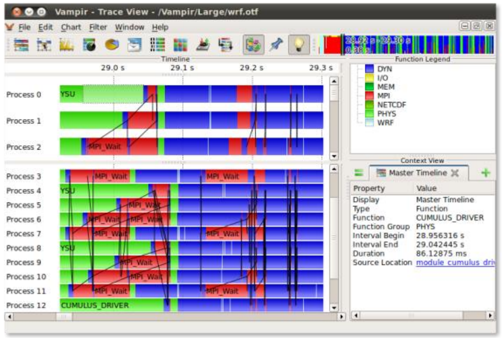


Performance Analysis Tools



Event-based Performance Analysis Workflow





Application

Measurement Tool

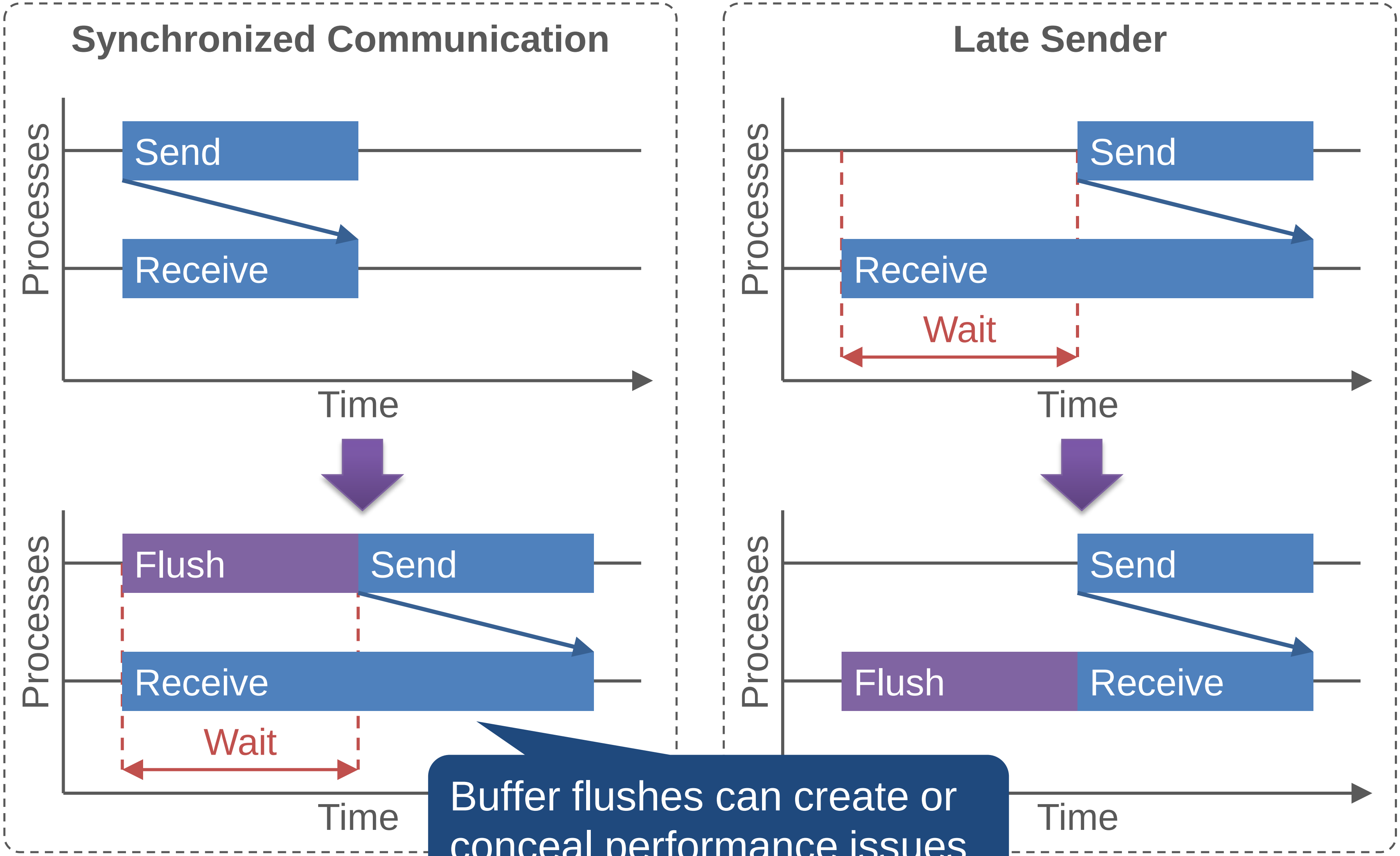
One of the biggest issues: bias due to intermediate memory buffer flushes

Analysis

Analysis Tool

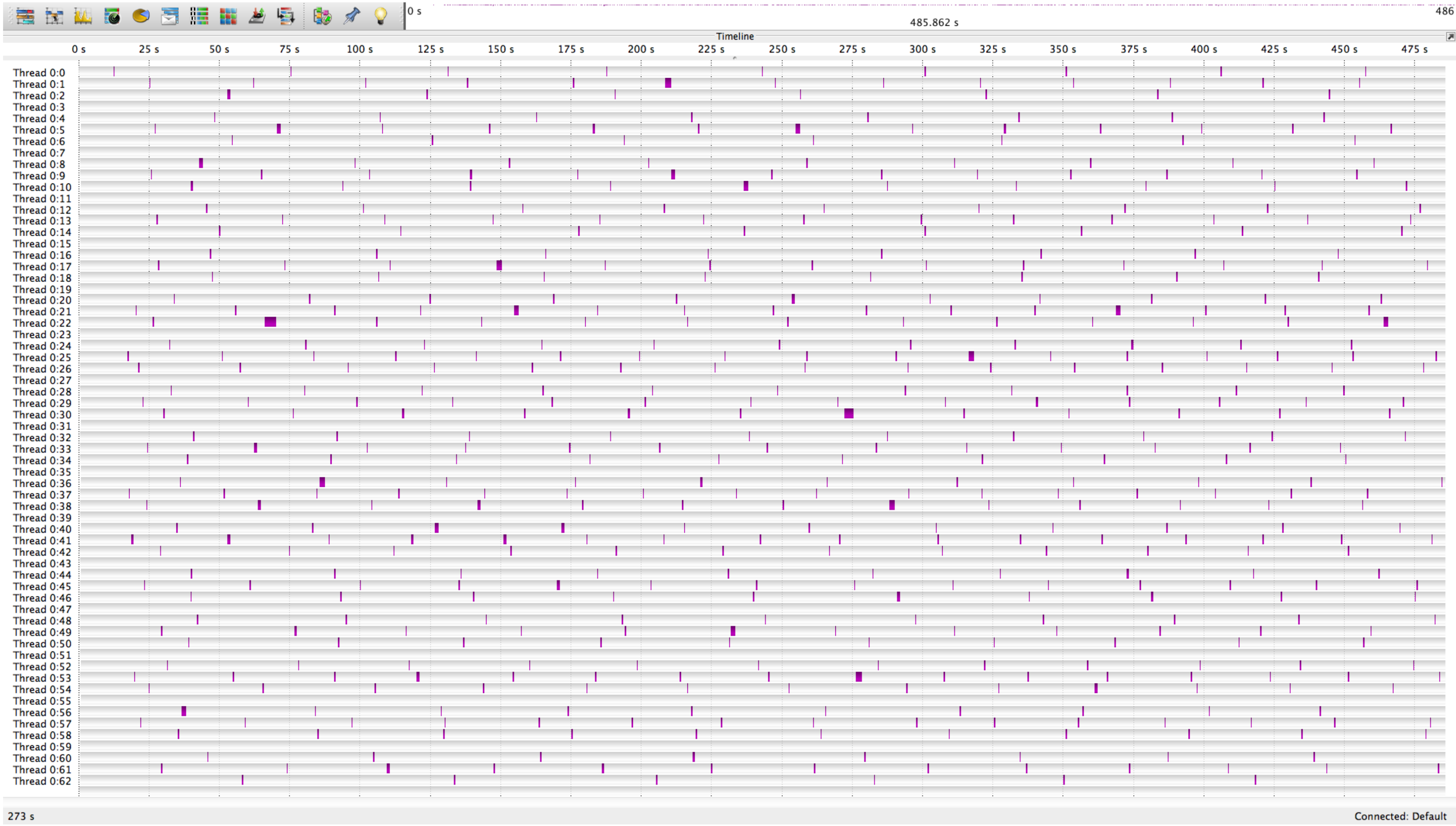
File System

Intermediate Memory Buffer Flushes



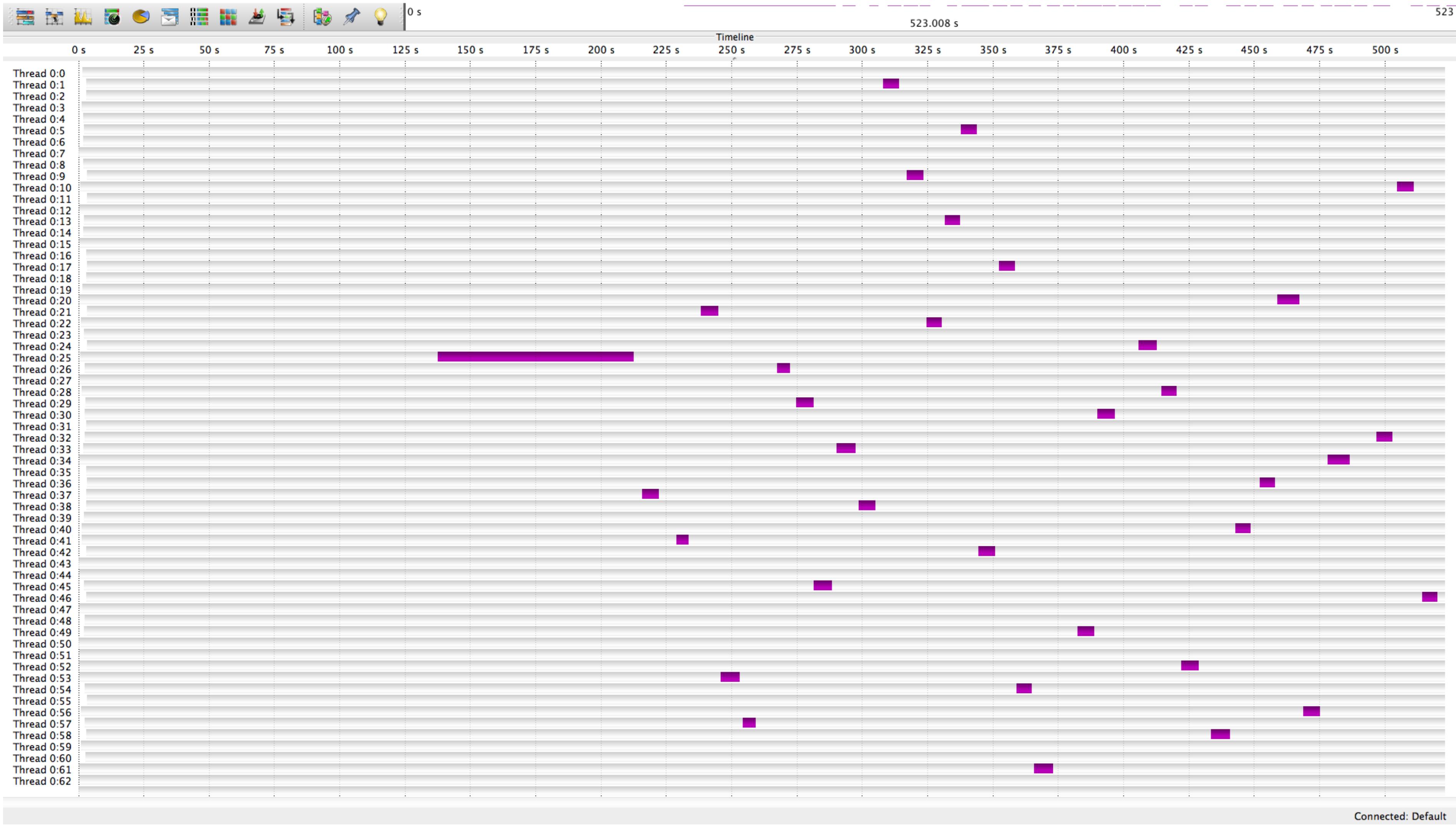
Buffer flushes can create or conceal performance issues

Intermediate Memory Buffer Flushes



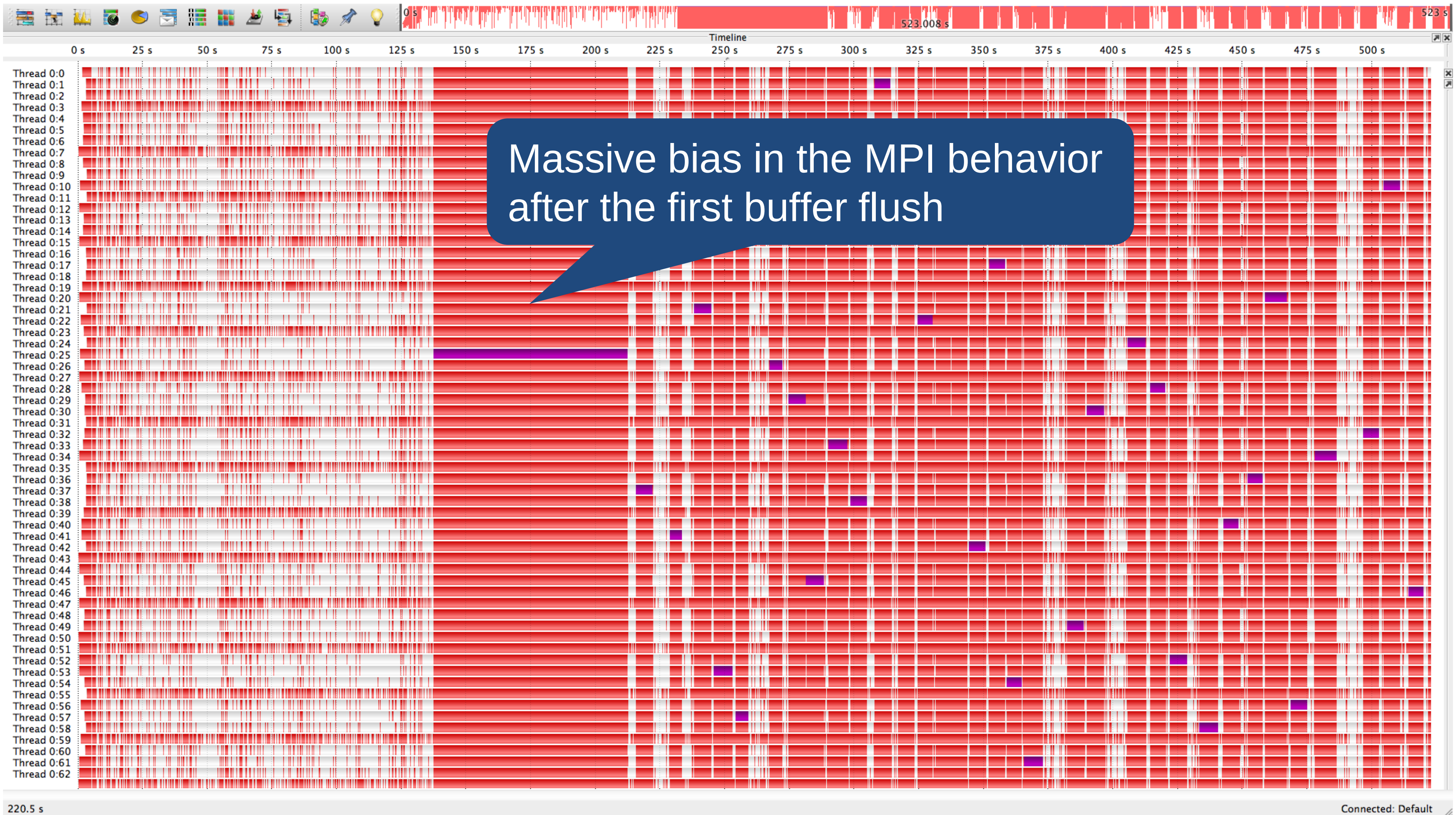
- Buffer flush, buffer size: 100 MiB

Intermediate Memory Buffer Flushes (2)



- Buffer flush, buffer size: 1 GiB

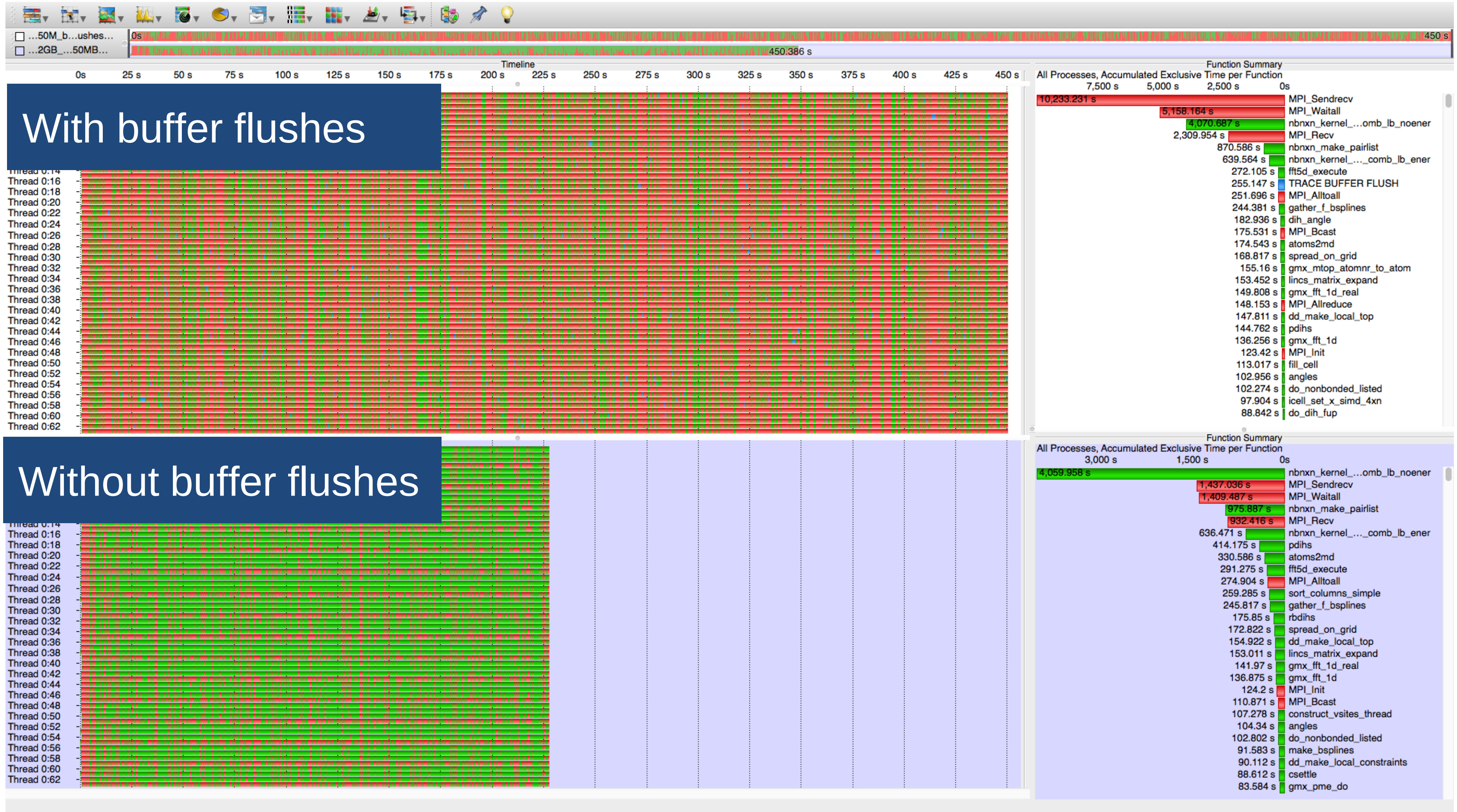
Intermediate Memory Buffer Flushes (3)



- Buffer flush, buffer size: 1 GiB
- MPI

Barcelona Supercomputing Center
Centro Nacional de Superco

Measurement Bias



MPI-only Tracing

Application	Trace size (per process)	
	OTF2	MPI-only
gromacs	1.7 GB	9.8 MB
cosmo-specs	1.5 GB	80 KB
3dbox	919 MB	8.8 MB
pipe	817 MB	8.5 MB
colloid	900 MB	12 MB
lennard-jones	1.8 GB	690 kB
rigid	709 MB	680 kB

?

- MPI-only tracing drastically reduces trace size
- Communication events lose their context in the application behavior

State-of-the-art

Complete trace
May contain large bias and falsified information

?

MPI-only trace
Allows correct but context-free communication analysis

Hybrid Measurement:
Detailed tracing for MPI and adaptive sampling for computation
Provide complete MPI communication and reduce the remaining application behavior to fit into a single buffer

Prerequisites for Hybrid Measurement

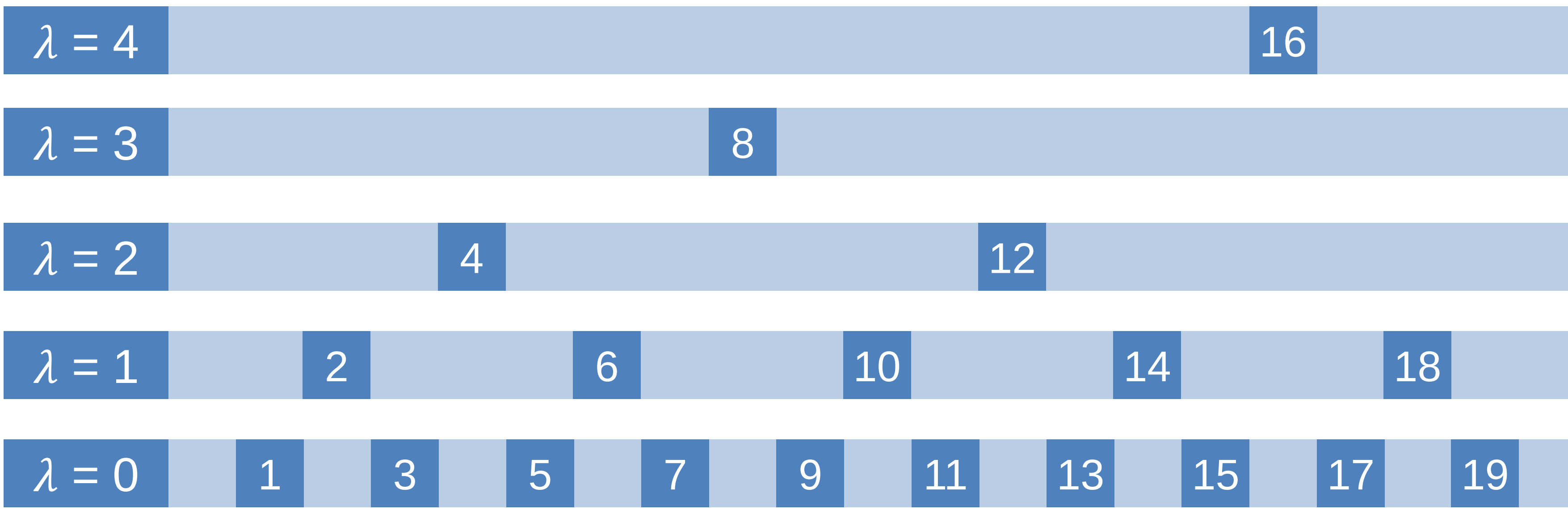
1. Change the sampling frequency from the tool when the memory buffer is full
   - Pre-Flush callback it OTF2
   - Adaption of the sampling frequency done by the tool
2. Remove already stored samples from the memory buffer
   - Utilize the Hierarchical Memory Buffer in OTFX
   - Allows efficient removal of hierarchically sorted data e.g. events

- Provide hierarchical order to samples
- Use hierarchy based on order of occurrence

Distribution Function

- Distribution function $\lambda : \mathbb{N} \rightarrow \mathbb{N}$ to map each sample to the according level based on the order of occurrence n:

$$\lambda(n) = max\{\, p \in \mathbb{N} \mid n \equiv 0 \; mod \; 2^p \}$$



Distribution Function (2)

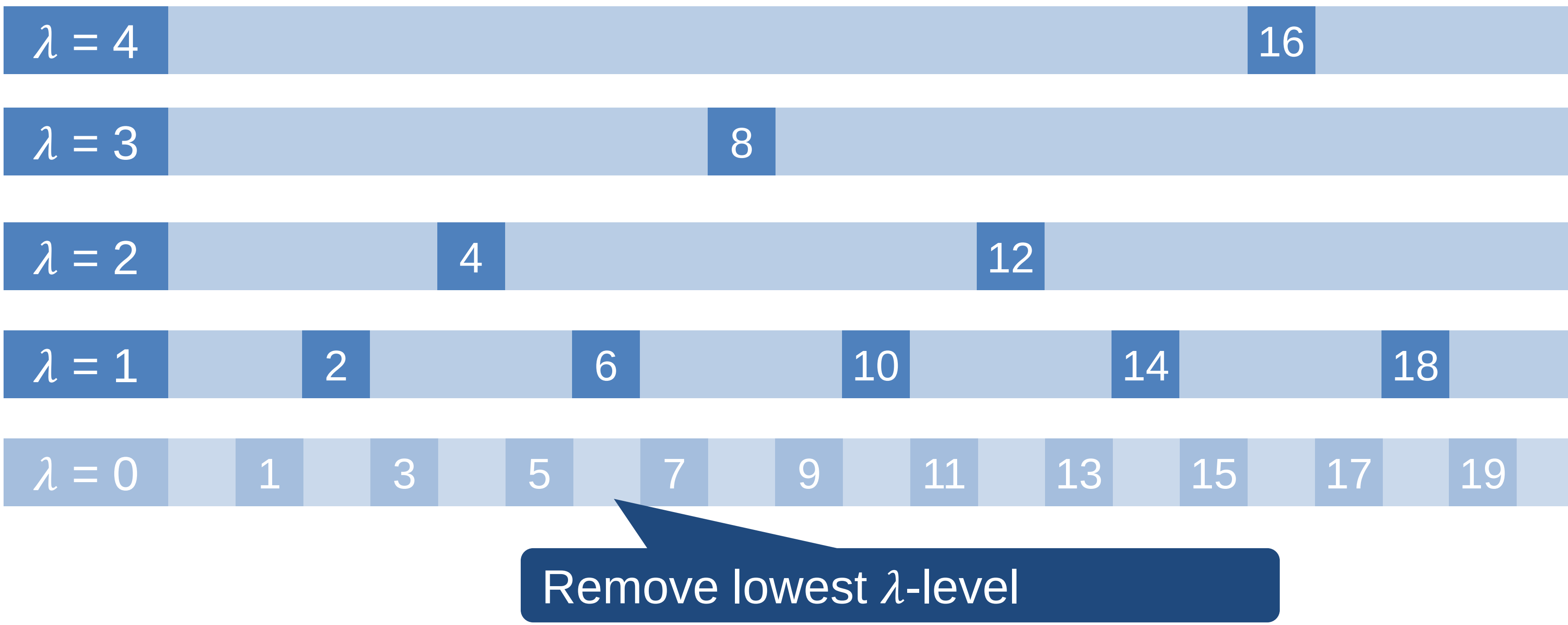
- Distribution function $\lambda : \mathbb{N} \rightarrow \mathbb{N}$ to map each sample to the according level based on the order of occurrence n:

$$\lambda(n) = max\{\, p \in \mathbb{N} \mid n \equiv 0 \; mod \; 2^p\}$$

- Basic Properties
  - Each level λ_p contains every 2^{p+1}th sample
  - λ_0 contains every second sample
  - λ_1 contains every fourth sample
  - …
  - $\lambda_{max} = \lambda_{\lfloor \ln n \rfloor}$ contains one sample
- Further Properties
  - The interval of levels $[\lambda_p, \infty)$ contains every 2^pth sample
  - $[\lambda_1, \lambda_{max}]$ contains every second sample

Distribution Function – Removal Operation

- Removal operation: remove lowest λ-level containing every second sample
- After: the new lowest λ-level contains every second sample
- Reduce the future sampling frequency by half



Distribution Function – Efficient Calculation

- Computation of max-notation can be very costly for large n
- Since any natural number n can be uniquely decomposed in powers of two:

$$n = \sum_{p \in \mathbb{N}} \alpha_p 2^p \; and \; \alpha_p \in \{0,1\}$$

- The distribution function can also be expressed as:

$$\lambda(n) = min\left\{p \in \mathbb{N} \,\middle|\, n = \sum_{p \in \mathbb{N}} \alpha_p 2^p \; and \; \alpha_p \in \{0,1\}\right\}$$

- The min-notation equals the binary representation of integers
- λ is equal to the number of trailing zeros

Distribution Function – Efficient Calculation (2)

- The number of trailing zeros can be efficiently computed using de Bruijn sequences:
- A de Bruijn sequence is a cyclic sequence in which every possible string of length n out of an alphabet A occurs exactly once as a substring
- Example:
  - Alphabet A = {0, 1}, n = 2
  - All possible substrings of length n = 2 {00, 01, 10, 11}
  - 0011 is de Bruijn sequence since every substring occurs exactly once
    0011, 0011, 0011, 0011

Distribution Function – Efficient Calculation (3)

```
unsigned int input;
int lamdba;
static const int lookup[32] =
{
    0, 1, 28, 2, 29, 14, 24, 3,
    30, 22, 20, 15, 25, 17, 4, 8,
    31, 27, 13, 23, 21, 19, 16, 7,
    26, 12, 18, 6, 11, 5, 10, 9
};
lambda = lookup[((input & -input)*0x077CB531U) >> 27];
```

C. E. Leiserson, H. Prokop, and K. H. Randall, “Using de Bruijn Sequences to Index a 1 in a Computer Word,” 1998.

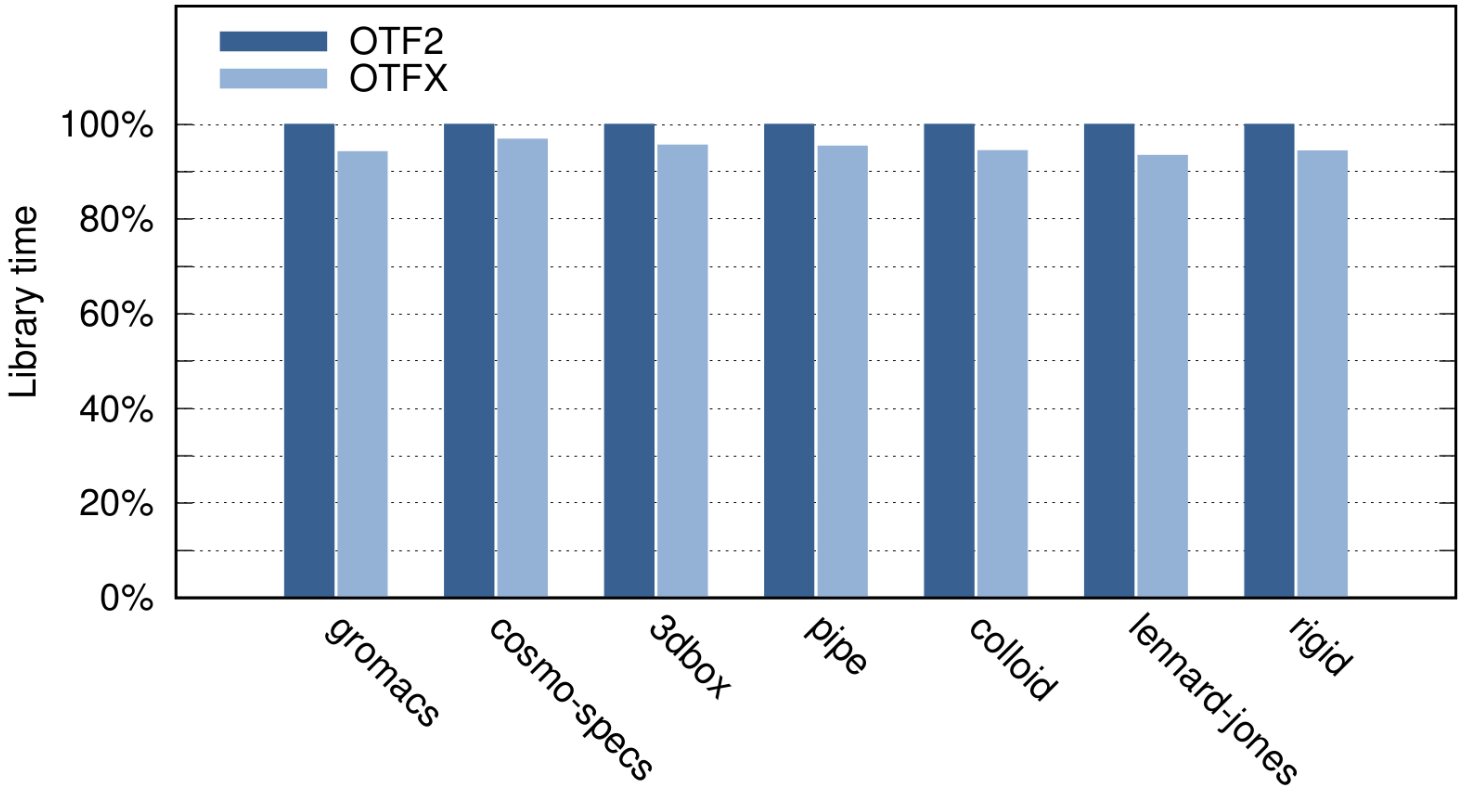
Lookup of trailing zeros

Right shift by 27

Get least significant 1 bit

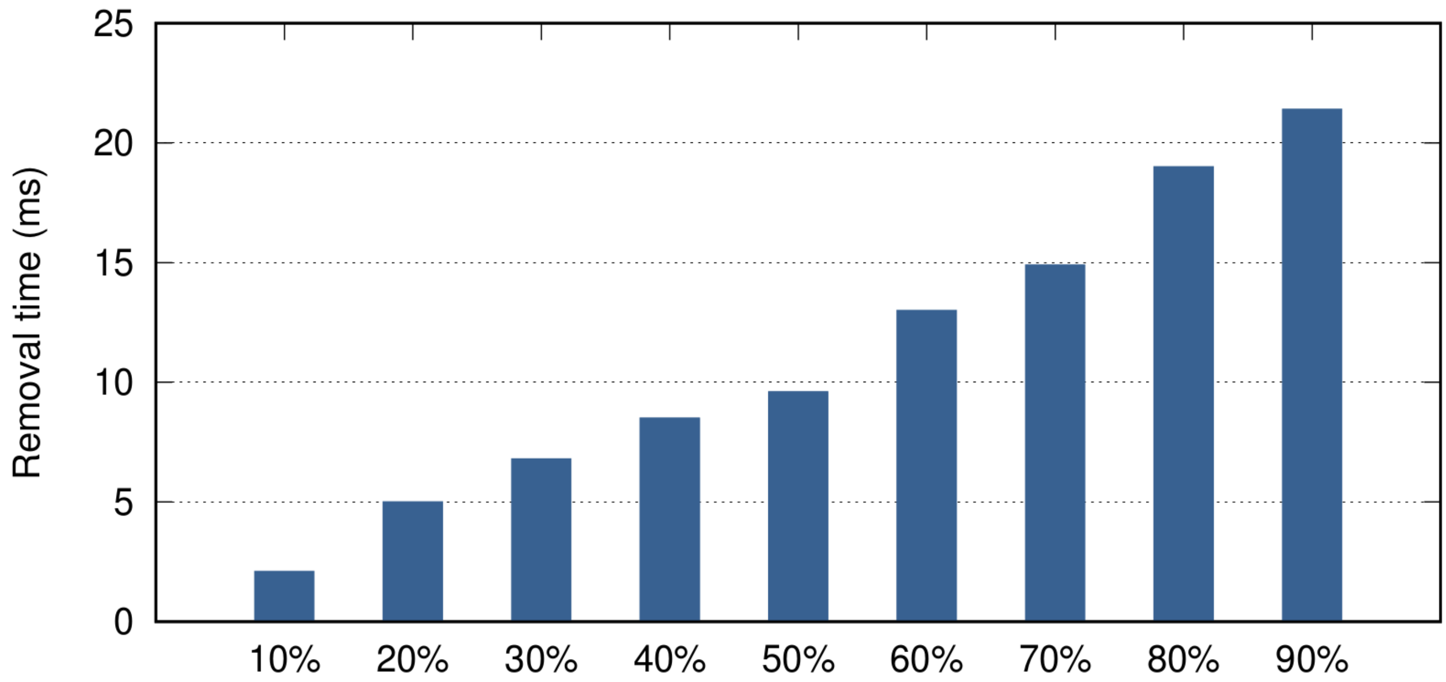
De Bruijn sequence: unique pattern into the 5 highest bits

Evaluation: Runtime Overhead



- Trace replay to ensure equal input data for both libraries
- In average 5.1% faster than OTF2
- Multiply & lookup for λ takes about 2 – 2.5 cycles

Evaluation: Removal Overhead



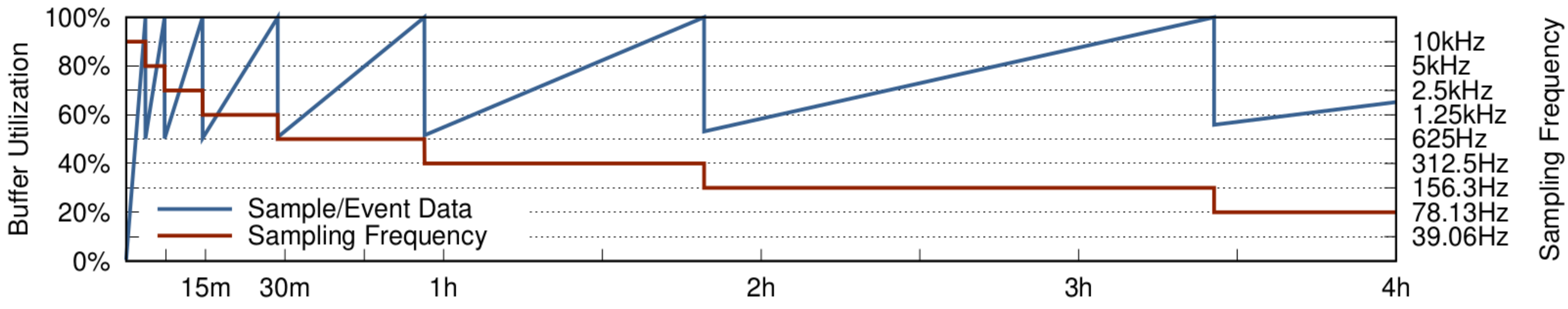
- Removal operation scales linear with the amount of data to be removed
- Single removal operation maximum of 50% (λ distribution)
- Maximum overhead 10ms for 100MB buffer
- Noticeable but much smaller than buffer flush with 500-600ms

Feasibility and Use Cases

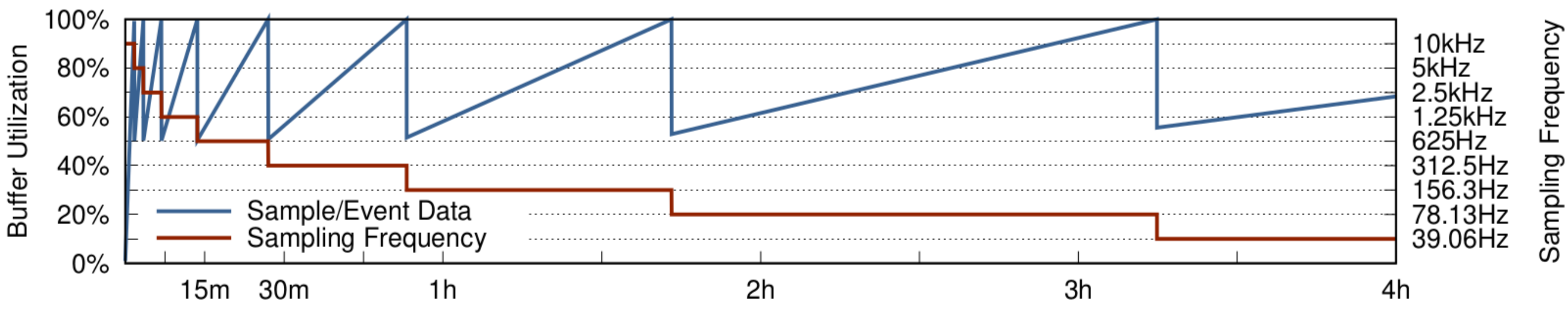
- Model based on:
  - Initial sampling frequency f_s
  - Measurement duration t_m
  - Amount of collect data per sample d_s
  - Rate of other events (e.g. MPI events) r_e
- Model parameters:
  - f_s = 10kHz
    - Score-P and Extrae use 100Hz and 20 Hz
    - Good trade of between accuracy and overhead
  - $d_s \in$ {48B, 102B}
    - Based on records in OTF2 with 2/8 counters
  - $r_e \in$ {1kB/s, 10kB/s}
    - Oriented on the behavior of Gromacs and Cosmo-Specs+FD4
  - t_m is variable

Feasibility and Use Cases (2)

- Scenario A: f_s = 10kHz, d_s= 48B, r_e = 1kB/s



- Scenario B: f_s = 10kHz, d_s= 102B, r_e = 1kB/s



Conclusion

- Novel approach to automatically adapt the sampling frequency to the given buffer space
  - Applicable for sampling-based and hybrid event/sample-based monitors
  - Keeps MPI with high accuracy and detail and adapts the sampling rate
  - Provides much higher accuracy than existing approaches with fixed sampling frequency in many use cases
  - Frees the user from estimating or guessing of the optimal sampling rate
- Prototype implementation in OTFX
  - On average 5.1% less overhead than OTF2
  - Maximum of 10ms for removal vs. 500-600ms for buffer flush
  - Automatically selects suitable sampling frequency