

Minisymposium

7 & 8 July 2021

11:00 – 13:00 CEST



Performance Optimisation and Productivity for EU HPC Centres of Excellence (and European parallel application developers preparing for exascale)

Marta Garcia-Gasulla (Barcelona Supercomputing Center)

Brian Wylie (Jülich Supercomputing Centre)

- Imminent exascale computer systems challenge application developers
 - beyond commonplace execution performance and scaling inefficiencies, that also limit productivity of all parallel applications
- EU has funded HPC Centres of Excellence to prepare applications
 - including transversal Performance Optimisation and Productivity (POP) CoE, to support the other CoEs and the wider community of European developers

EU HPC Centres of Excellence



<https://www.hpccoe.eu>





EU HPC Centre of Excellence Performance Optimisation & Productivity

Brian Wylie (Jülich Supercomputing Centre)

EU H2020 Centre of Excellence (CoE)



1 October 2015 – 31 March 2018
1 December 2018 – 30 November 2021

Grant Agreement No 676553 and No 824080



Performance Optimisation and Productivity

EU H2020 Centre of Excellence in HPC

- Promotes best practices in parallel programming
 - Improving parallel software can add a lot of value: reduced expenditure, faster results, novel solutions
 - **POP Methodology** - a systematic approach to performance analysis and optimization building a quantitative view of parallel application execution behaviour
- Free services for all European (EU/EEA and associate countries) academic and industrial codes and users
 - Suggestions on improving code performance, described in a **Performance Assessment**
 - Practical help with code refactoring and optimization via prototyping in a **Proof of Concept**



- A team with
 - Excellence in performance tools and tuning
 - Excellence in programming models and practices
 - R & D background in real academic and industrial use cases

For further information, visit:  <https://www.pop-coe.eu>

 pop@bsc.es

 [@POP_HPC](https://twitter.com/POP_HPC)

 youtube.com/POPHPC

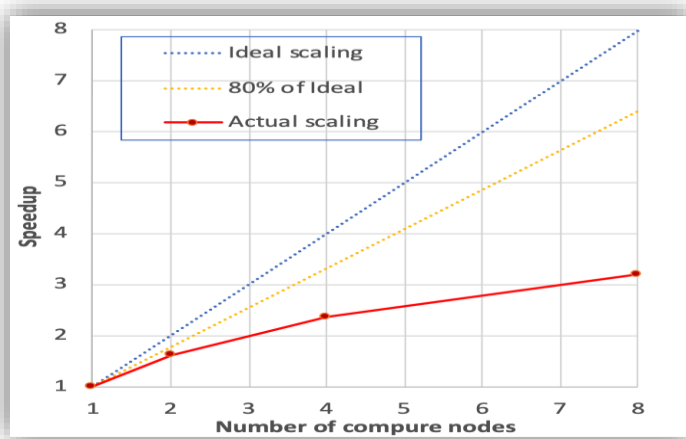


Parallel performance is hard to understand

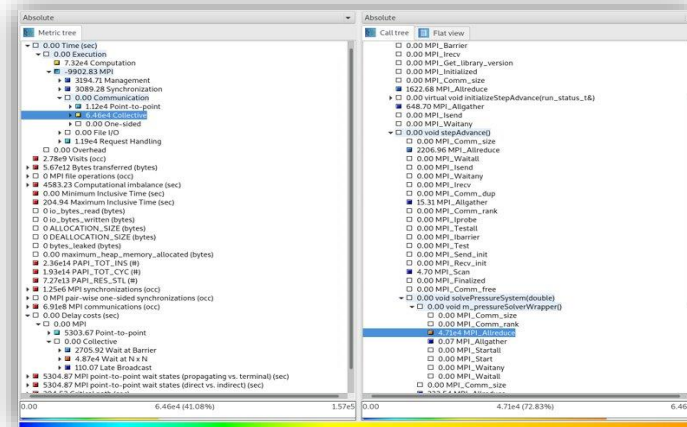


How do we measure the performance of our parallel programs?

- Traditional speed-up and efficiency plots?
- Profiling & tracing with performance tools?
 - Each technique is powerful, but potentially generates overwhelming amount of data



Speedup plot



Cube, performance metrics per routines/call path, data collected by Scalasca/Score-P



Paraver, timeline view of program execution, data collected by Extrae

Difficult to know where to start and what to look for

Main Problem: Lack of quantitative understanding of the actual execution behaviour of a parallel application



POP performance tools



Open-source tools

- Extrae (tracing), Paraver (analysis & visualisation) & Dimemas (modelling)
 - <https://tools.bsc.es>
- Score-P (profiling and tracing), Scalasca (extended analyses) & CUBE (presentation)
 - <https://www.scalasca.org> <https://www.score-p.org>
- MAQAO: synthetic reports and hints with a focus on processor core performance
 - <http://www.maqao.org>
- PyPOP: automated generation of POP metrics from Extrae traces
 - <https://github.com/numericalalgorithmsgroup/pypop>

MUST (MPI correctness) &
Archer (OpenMP correctness)
• <https://hpc.rwth-aachen.de/must>

For more help on how to use these tools and calculate the POP metrics

- See the POP website learning material & online training
 - <https://pop-coe.eu/further-information/learning-material>
 - <https://pop-coe.eu/further-information/online-training>

Other tools can also be used





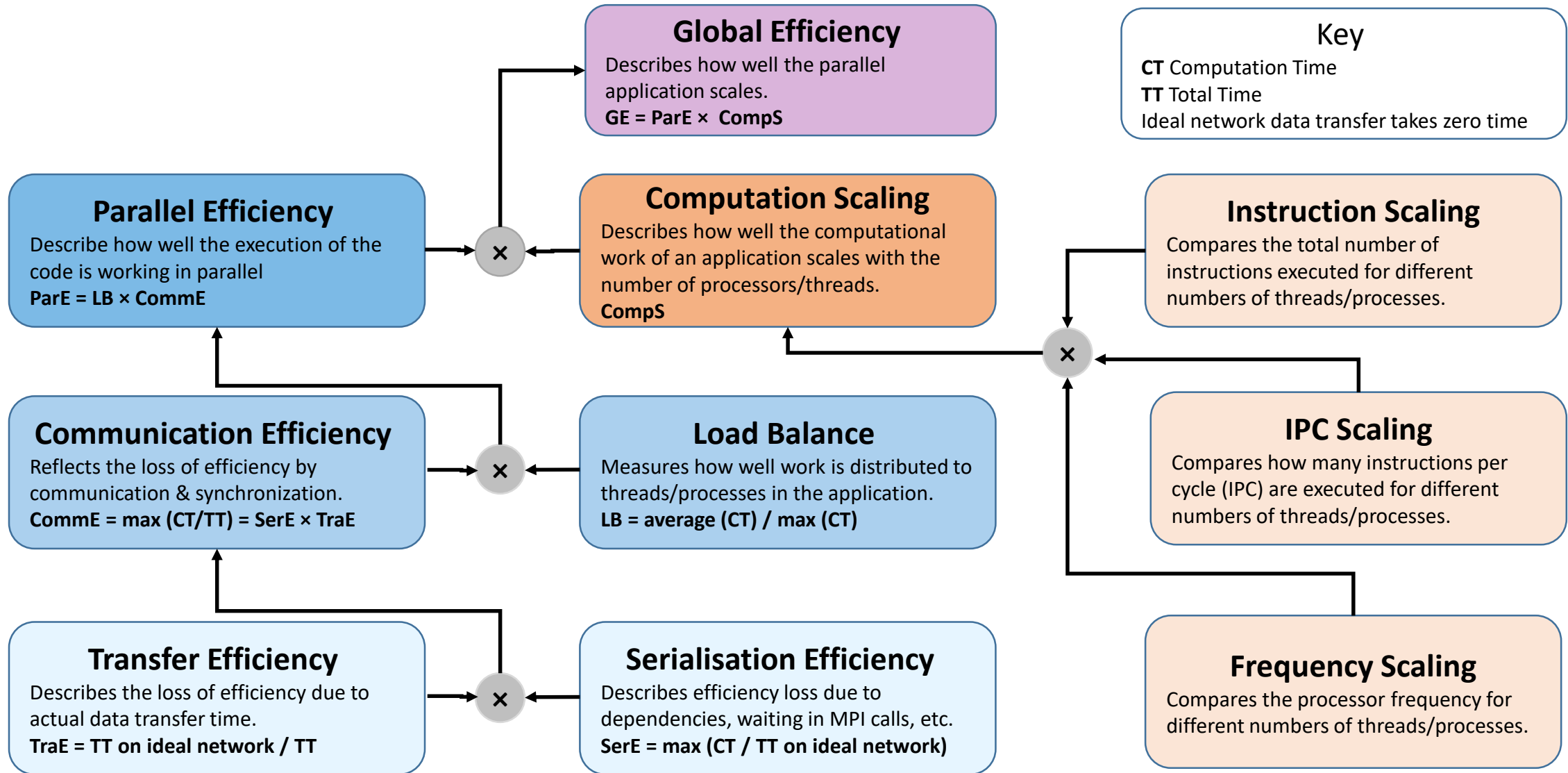
Simple but extremely powerful idea

- Devise a simple set of performance metrics using values easily obtained from profile & trace measurement data
- Where low efficiency values indicate causes of poor parallel performance for a Focus of Analysis (FoA)

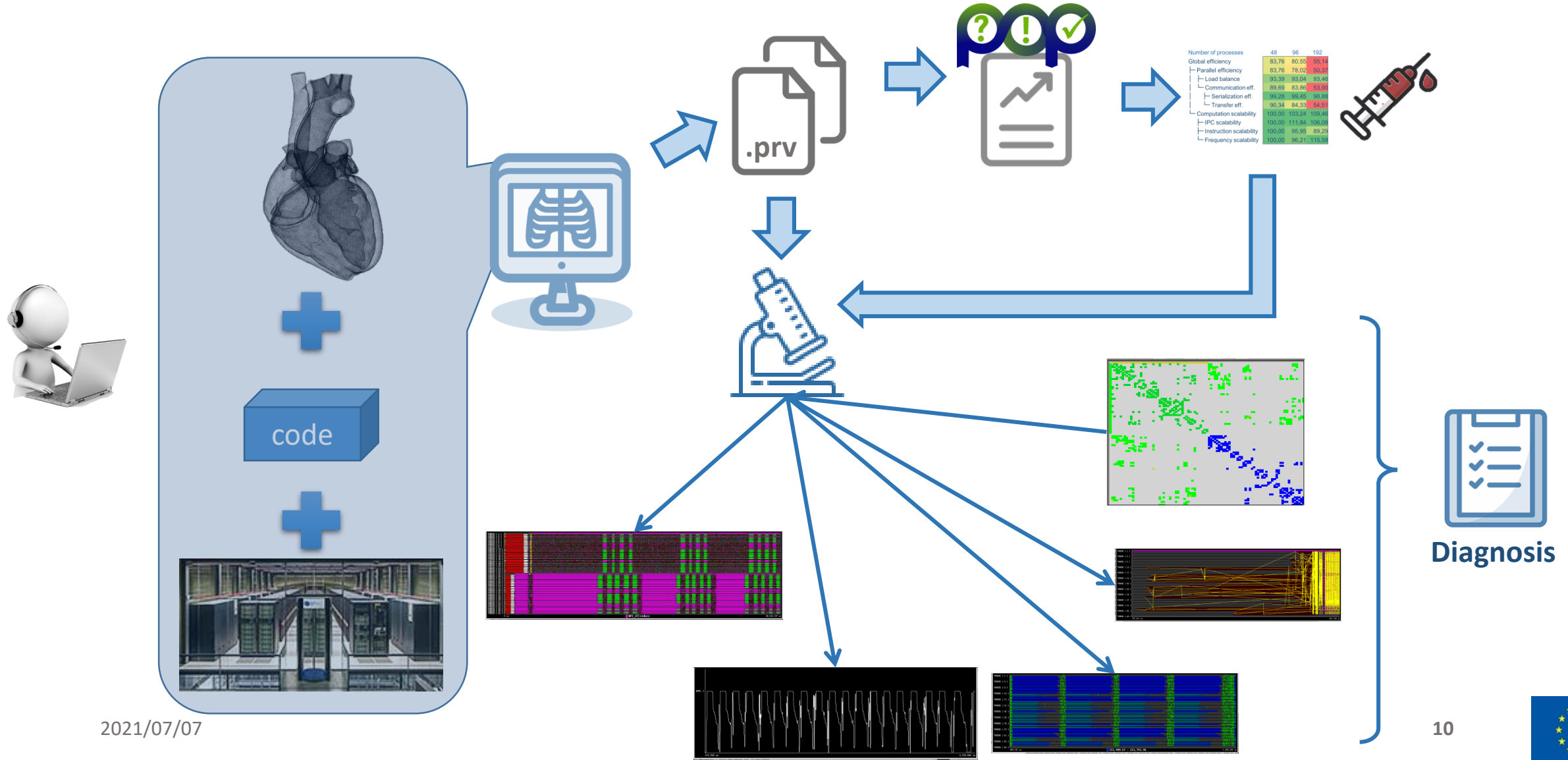
These metrics then are used to understand

- What are the causes of poor performance
- Where it occurs in the application execution
- Additionally, the metrics provide a common ground for discussing performance issues
 - Between developers, users and analysts

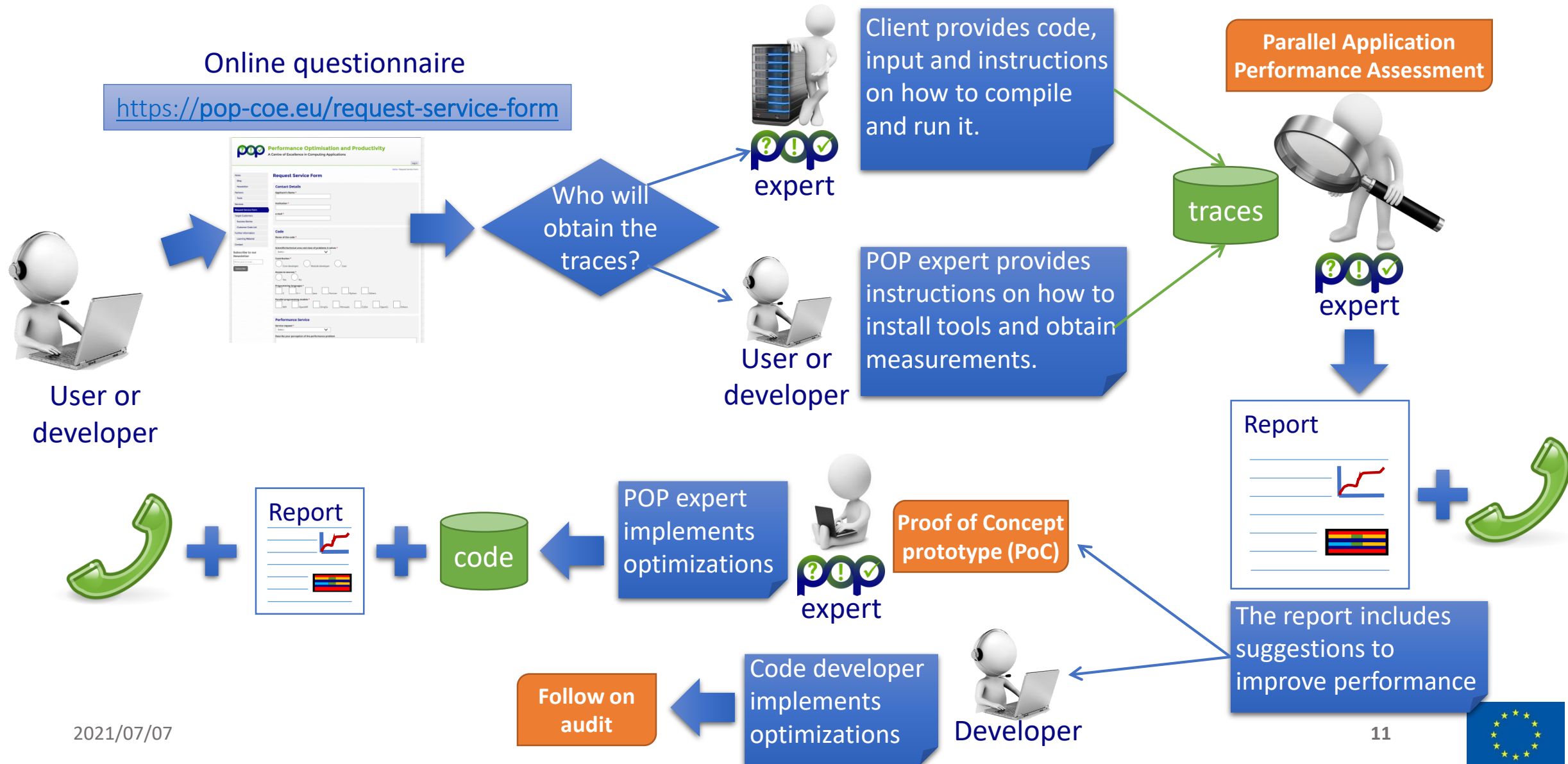
POP Scaling Efficiency Metrics (MPI)



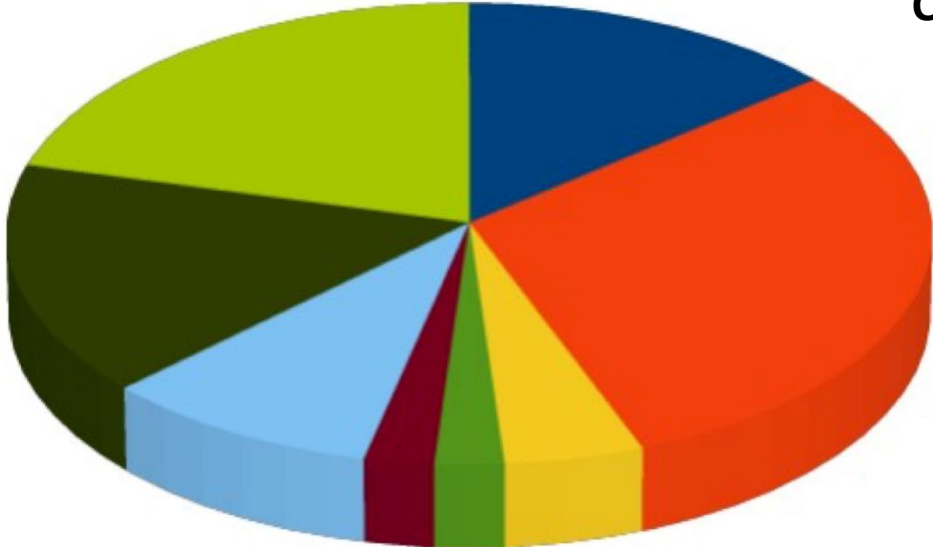
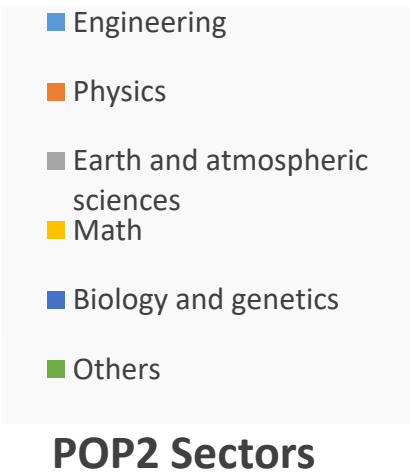
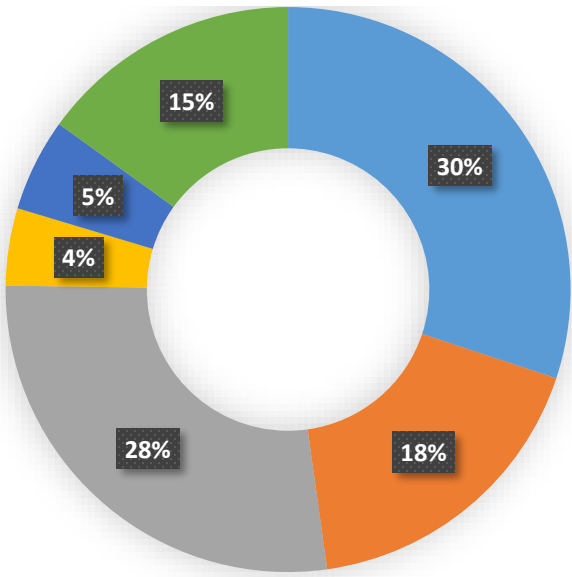
Performance assessment



POP services

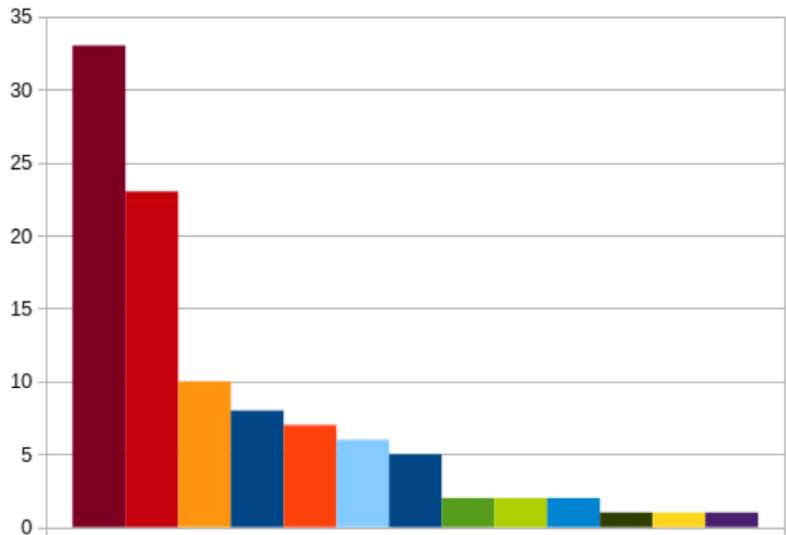


POP2 Services & HPC Codes



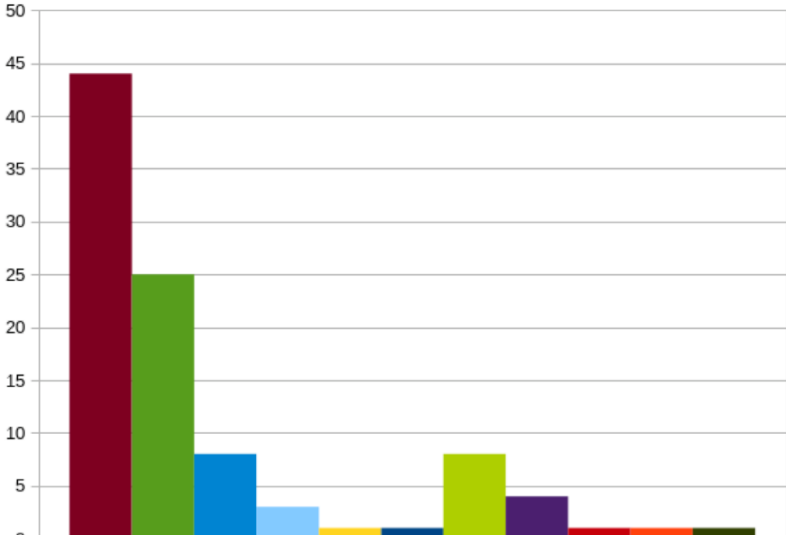
Code License

- BSD
- GPL / LGPL
- Apache
- no license
- Public
- Other
- internal use
- Commercial



Programming Languages

- C++
- Fortran
- C++, Python
- C
- C, Fortran
- C++, Fortran
- Python
- C, Python
- C++, Fortran, Python
- Fortran, Python
- C++, Fortran, Perl
- C, Java
- C++, Fortran, Python, Java



Parallel Programming Models
















- MPI
- MPI+threads
- MPI+threads+accel
- MPI+accel
- MPI+TBB
- MPI+TBB+accel
- threads
- threads+accel
- Origami HPC
- Celery
- WDL

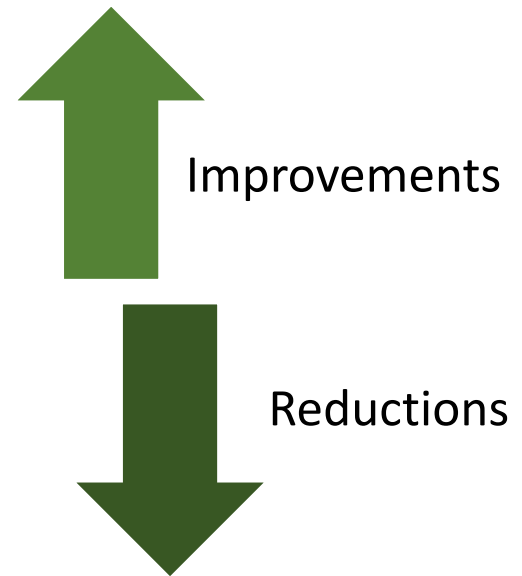


Some Success Stories



- See [⇒ https://pop-coe.eu/blog/tags/success-stories](https://pop-coe.eu/blog/tags/success-stories)

-  POP for Astronomy - **40% Reduction** in Execution Time for the PIERNIK Code
-  Performance **Improvements by More Than 30%** and a Data Race Fixed for CalculiX Code
-  **588x and 488x Execution Time Speedups** of a Volcanic Hazard Assessment Code
-  Vampire Magnetic Materials code **sped up by 46%**
-  Improvements in Shearwater Reveil seismic processing code of **up to 44% runtime reduction**
-  **Performance Improvements** for SCM's ADF Modeling Suite
-  **3x Speed Improvement** for zCFD Computational Fluid Dynamics Solver
-  **25% Faster time-to-solution** for Urban Microclimate Simulations
-  **2x performance improvement** for SCM ADF code
-  Proof of Concept for BPMF leads to around **40% runtime reduction**
-  POP audit helps developers **double their code performance**
-  **10-fold scalability improvement** from POP services
-  POP performance study improves performance **up to a factor 6**
-  POP Proof-of-Concept study leads to **nearly 50% higher performance**
-  POP Proof-of-Concept study leads to **10X performance improvement** for customer



- Resources for co-design
 - Database of execution performance patterns and corresponding best practice
- Training & education
 - Conference tutorials, tuning workshops, monthly training event series
 - On-line training modules & webinars
 - Joint application performance analysis workshops with E-CAM & EoCoE
- Periodic performance assessment campaigns
 - Initial & follow-up rounds for 10 application codes of ChEESE
 - Initial round for 8 application codes of CoEC
- Application exascale readiness assessments
 - e.g., progression of problem sizes, computer systems (CompBioMed)

- Imminent exascale computer systems challenge application developers
 - beyond commonplace execution performance and scaling inefficiencies, that also limit productivity of all parallel applications
- EU has funded HPC Centres of Excellence to prepare applications
 - including transversal Performance Optimisation and Productivity (POP) CoE, to support the other CoEs and the wider community of European developers
- Invited sectorial CoEs and developers of flagship application codes
 - to review their progress and exascale readiness status of their codes
 - to report their experience engaging with POP CoE and its services
 - to suggest desirable service/support additions/improvements



Chair: Brian Wylie

- Best practice for efficient and scalable application performance
 - Marta Garcia-Gasulla (Barcelona Supercomputing Center)
- Outcome of joint EoCoE-POP performance evaluation workshops
 - Mathieu Haefele (Université de Pau et des Pays de l'Adour)
- POP and ChEESE – the audit of *SeisSol* for computational earthquake simulations with GPU-aware MPI communication for local time stepping
 - Ravil Dorozhinskii (Technische Universität München)
- Readyng *HemeLB* and *SCEMa* codes for exascale with POP and E-CAM Centres of Excellence
 - Peter Coveney (University College London & University of Amsterdam)





Chair: Marta Garcia-Gasulla

- Performance optimization and productivity of *Alya* towards exascale
 - Ricard Borrell (Barcelona Supercomputing Center)
- Status of *NEMO* scalability and inputs from POP assessment
 - Sebastien Masson (LOCEAN, Sorbonne Université)
- Materials design towards the exascale: porting electronic structure community codes to GPUs
 - Andrea Ferretti (CNR-NANO)
- Panel discussion



Chair: Brian Wylie

- Marta Garcia-Gasulla (POP : Barcelona Supercomputing Center)
- Mathieu Haefele (EoCoE : Université de Pau et des Pays de l'Adour)
- Ravil Dorozhinskii (ChEESE : Technische Universität München)
- Peter Coveney (CompBioMed : University College London & U. Amsterdam)
- Ricard Borrell (CoEC/ChEESE/CompBioMed/EoCoE/EXCELLERAT/RAISE : BSC)
- Sebastien Masson (ESiWACE : LOCEAN, Sorbonne Université)
- Andrea Ferretti (MaX : CNR-NANO)

- What challenges remain for your CoE (flagship) applications to be ready to effectively exploit (imminent) exascale computer systems?
 - Initially targeting one type of computer system or preparing for several types?
- What POP services have you found to be most beneficial?
 - How should POP services (including training) be extended/improved?



Best Practice for Efficient and Scalable Application Performance

Marta Garcia-Gasulla, BSC

EU H2020 Centre of Excellence (CoE)



Grant Agreement No 824080

1 December 2018 – 30 November 2021

- Between POP and POP2 more than 200 assessments performed.
- Only transversal CoE
- Assessed codes from almost all the other CoES
- Different fields, different sciences
- Common patterns and challenges



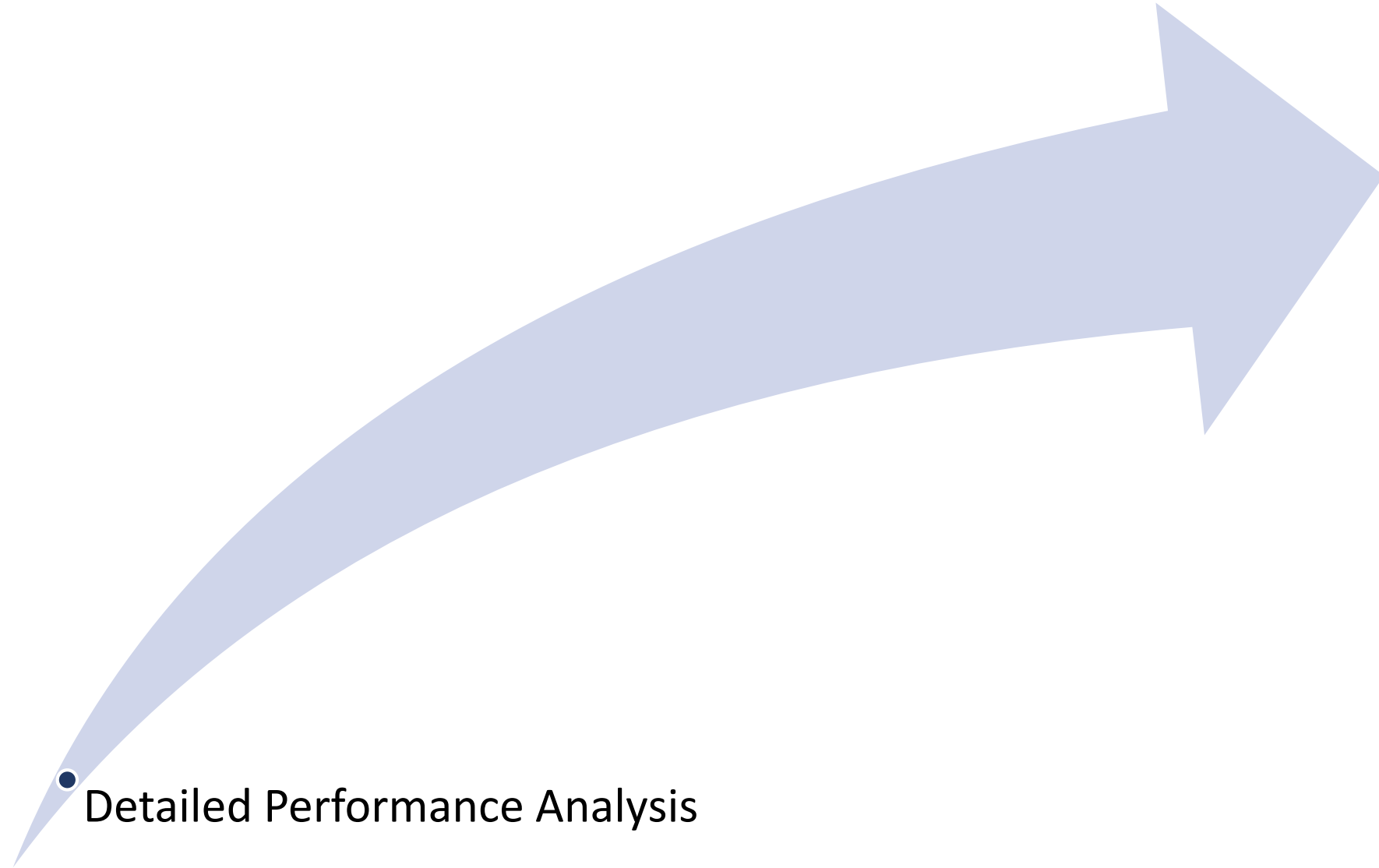
Disclaimer



- All the images of this presentation are taken from POP assessments or Proof-of-concepts of CoEs codes
- All the names and references have been intentionally removed



Step by step...



Detailed Performance Analysis

2021/07/07



Performance Analysis



- Forget about profiling and aggregated metrics

	Outside MPI	MPI_Send	MPI_Recv	MPI_Isend	MPI_Irecv	MPI_Waitall	MPI_Allreduce
Total	127,719.17 %	52.27 %	12,945.95 %	3,095.24 %	2,287.20 %	48,492.93 %	112,607.24 %
Average	41.58 %	0.02 %	4.21 %	1.01 %	0.74 %	15.79 %	36.66 %
Maximum	59.45 %	0.17 %	4.40 %	1.94 %	1.10 %	28.14 %	46.25 %
Minimum	36.44 %	0.01 %	1.90 %	0.56 %	0.46 %	6.41 %	26.50 %
StDev	2.37 %	0.01 %	0.09 %	0.13 %	0.08 %	3.52 %	3.43 %
Avg/Max	0.70	0.10	0.96	0.52	0.68	0.56	0.79

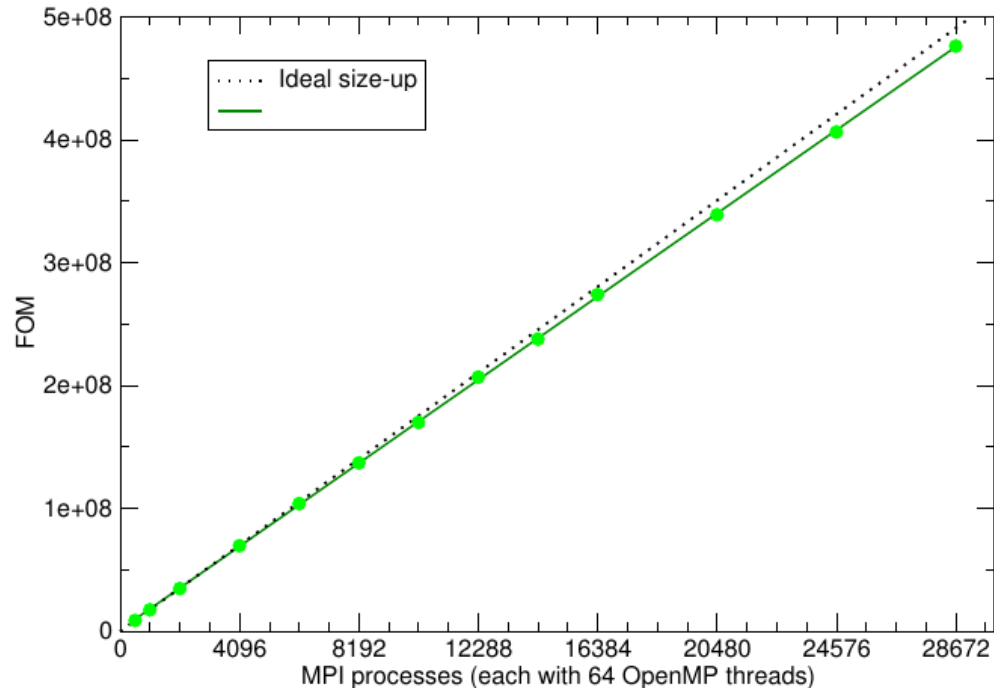
Nodes	16	32	64
Cores	768	1536	3072
Global efficiency	69	62	54
-Parallel efficiency	69	53	41
-Load Balance	83	78	69
-Communication efficiency	83	68	60
-Serialization efficiency	86	72	69
-Transfer efficiency	96	94	87
-Computational Scaling	100	118	131
-IPC Scaling	100	123	142
-Instruction Scaling	100	98	94
-CPU Frequency Scaling	100	98	98



Scaling the overhead



- Handle with care relative metrics and scalability



	Racks	$\frac{1}{2}$	1	2	4	8	16	28
Processor distribution		8x8x8	16x8x8	16x16x8	16x16x16	32x16x16	32x32x16	32x32x28
Processes		512	1024	2048	4096	8192	16384	28672
Threads		32768	65536	131072	262144	524288	1048576	1835008
Global efficiency		0.67	0.67	0.67	0.67	0.67	0.67	0.65
- Parallel efficiency		0.67	0.67	0.67	0.67	0.67	0.67	0.65
- - Load balance efficiency		0.96	0.96	0.96	0.96	0.96	0.96	0.95
- - Communication efficiency		0.70	0.70	0.70	0.70	0.70	0.70	0.69
- - - Serialization efficiency		0.70	0.70	0.70	0.70	0.70	0.70	0.69
- - - Transfer efficiency		1.00	1.00	1.00	1.00	1.00	1.00	1.00
- Computation efficiency		1	1.00	1.00	1.00	1.00	1.00	0.97
- - Instructions scaling		1	1.00	1.00	1.00	1.00	1.00	1.00
- - IPC scaling		1	1.00	1.00	1.00	1.00	1.00	0.98



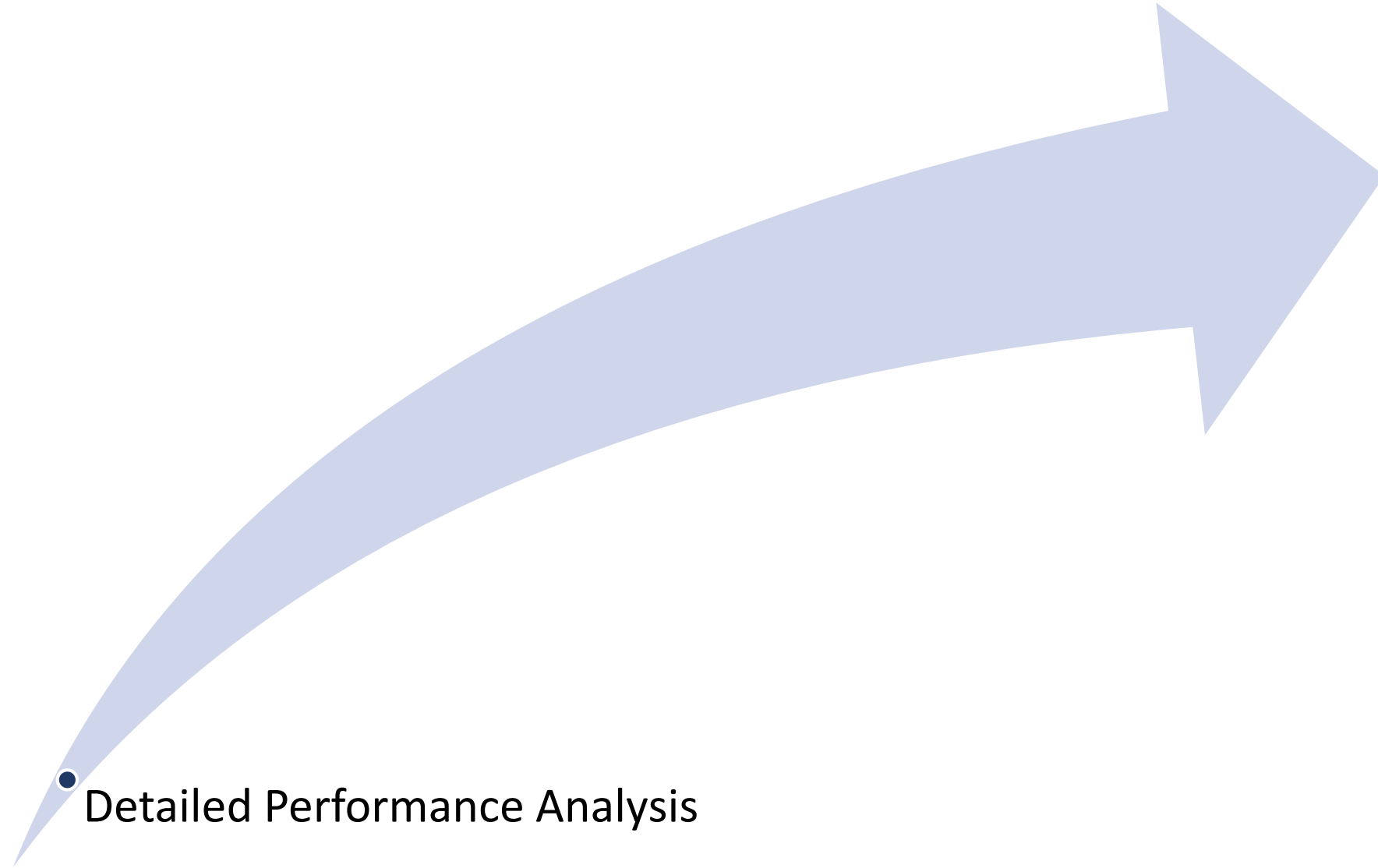
Performance Analysis



- Avoid profiling and be careful with relative metrics
- Performance analysts can help you with complementary view
- *If it ain't broke don't fix it!*
 - Analyze before implementing solutions
- Do not mask your symptoms



Step by step...

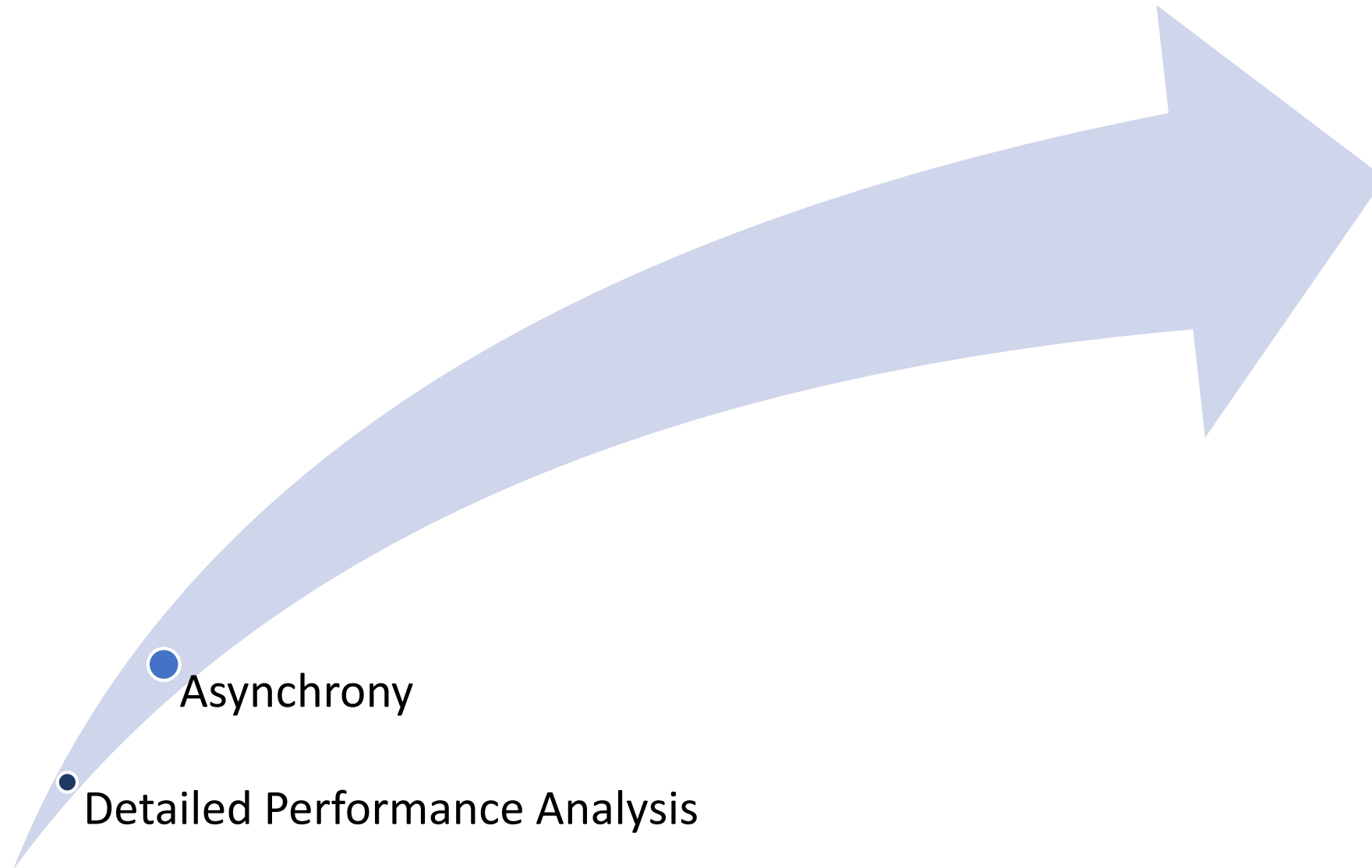


Detailed Performance Analysis

2021/07/07



Step by step...



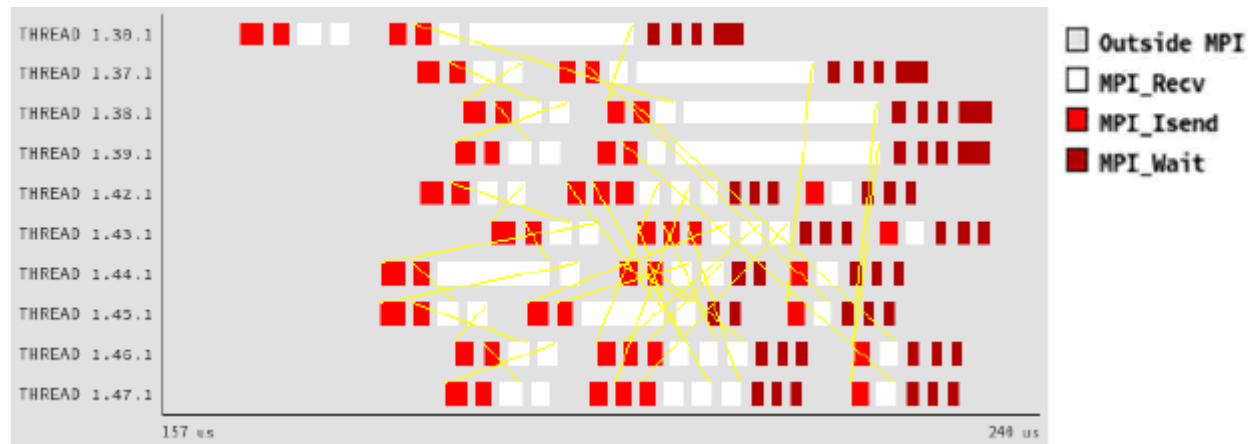
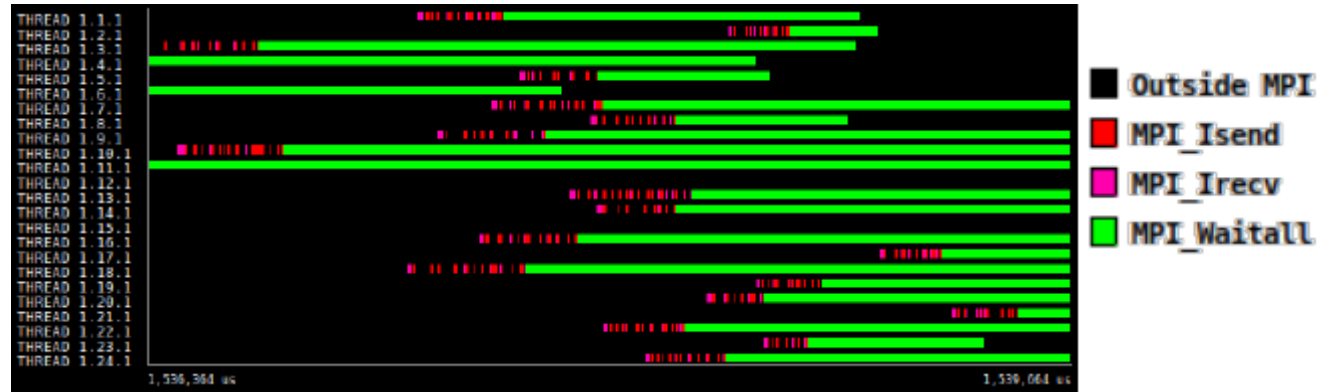
2021/07/07



Asynchrony



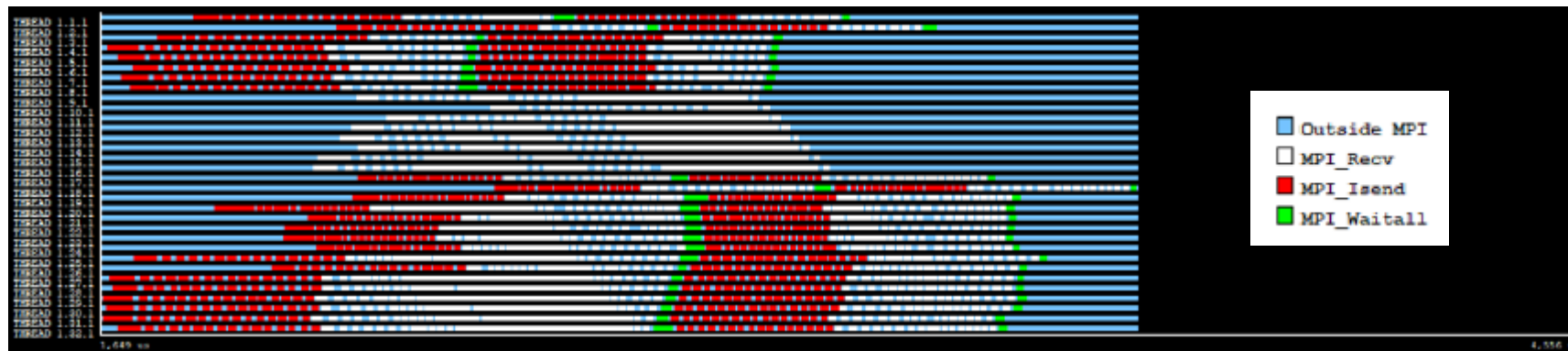
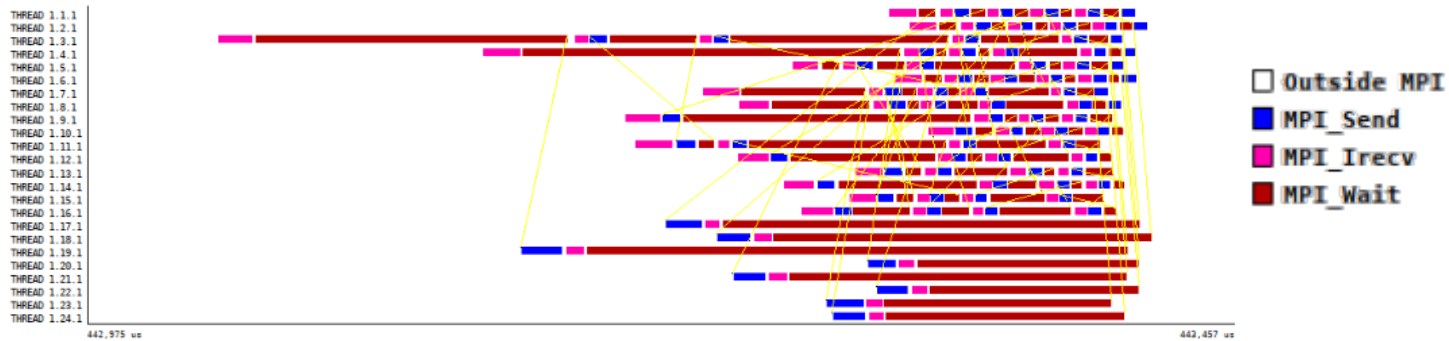
- In general its use is quite extended, with different patterns:



Asynchrony



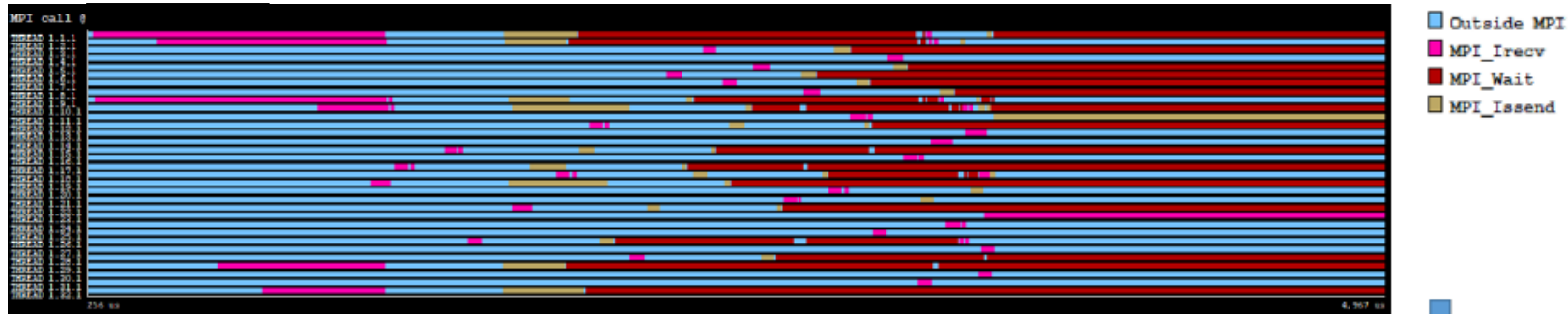
- Not always with great success...



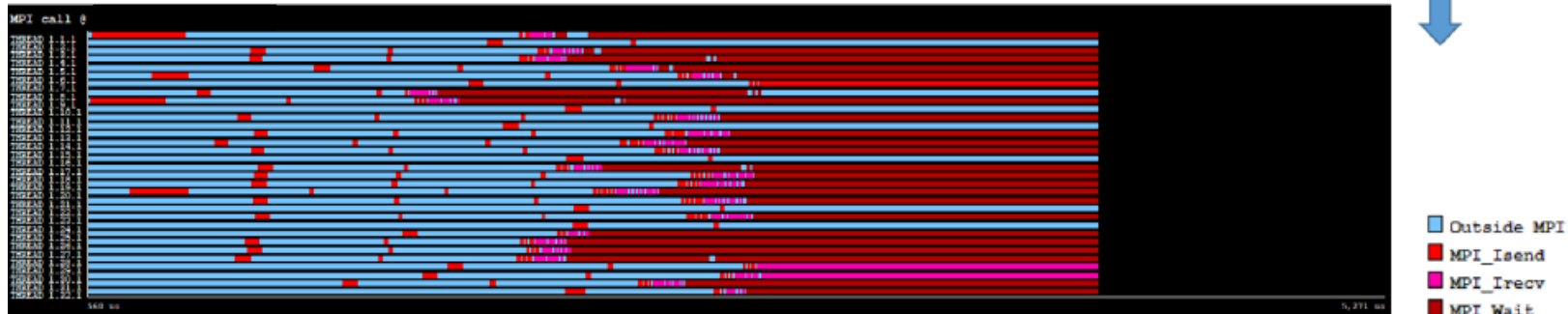
Asynchrony



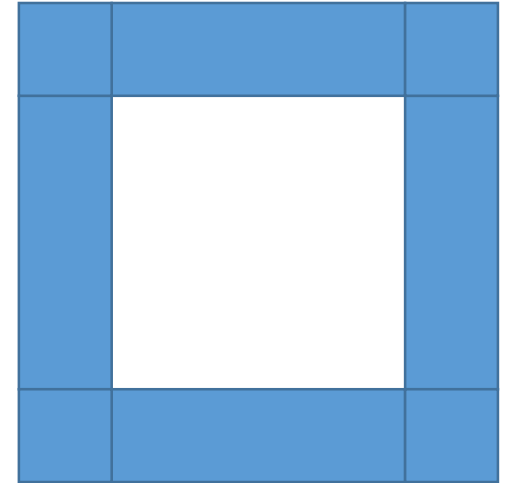
- But we are getting there...



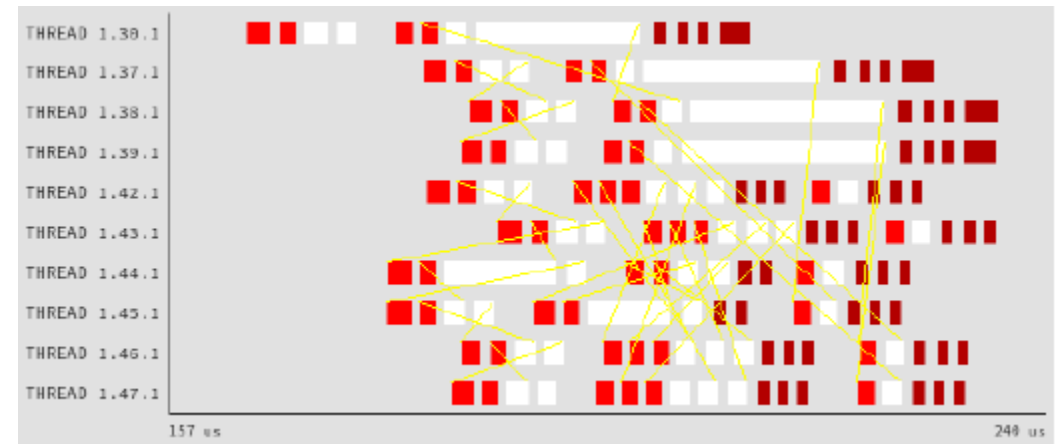
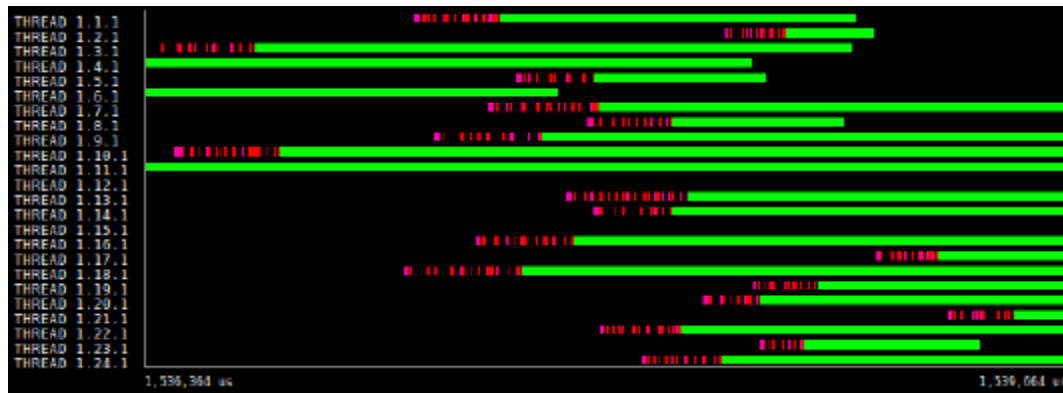
Proposed:



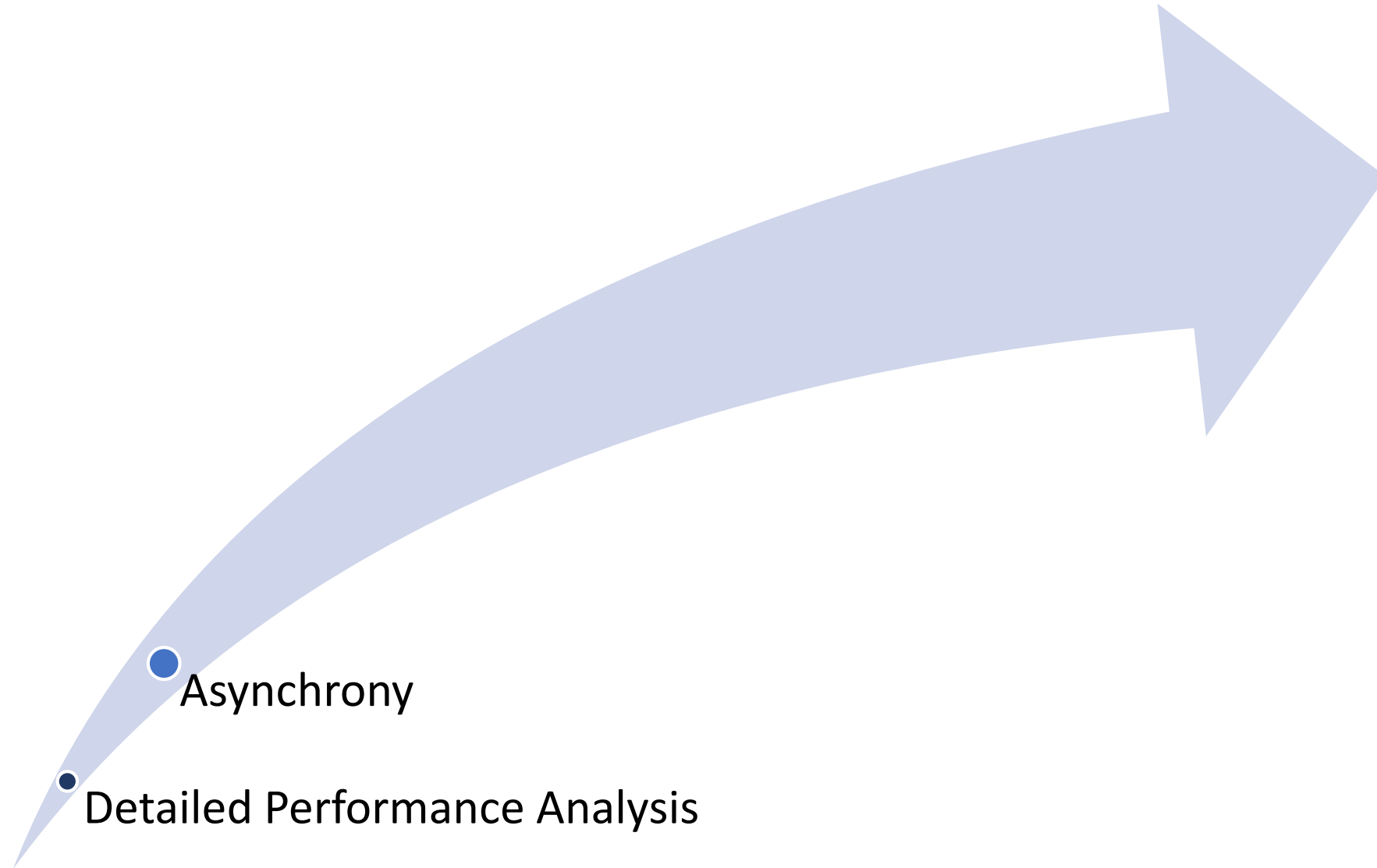
22% improvement



- Common pitfalls:
 - I'm using asynchronous calls, therefore communication is not a problem for me
 - I'm using asynchronous calls, therefore I am overlapping communication and computation



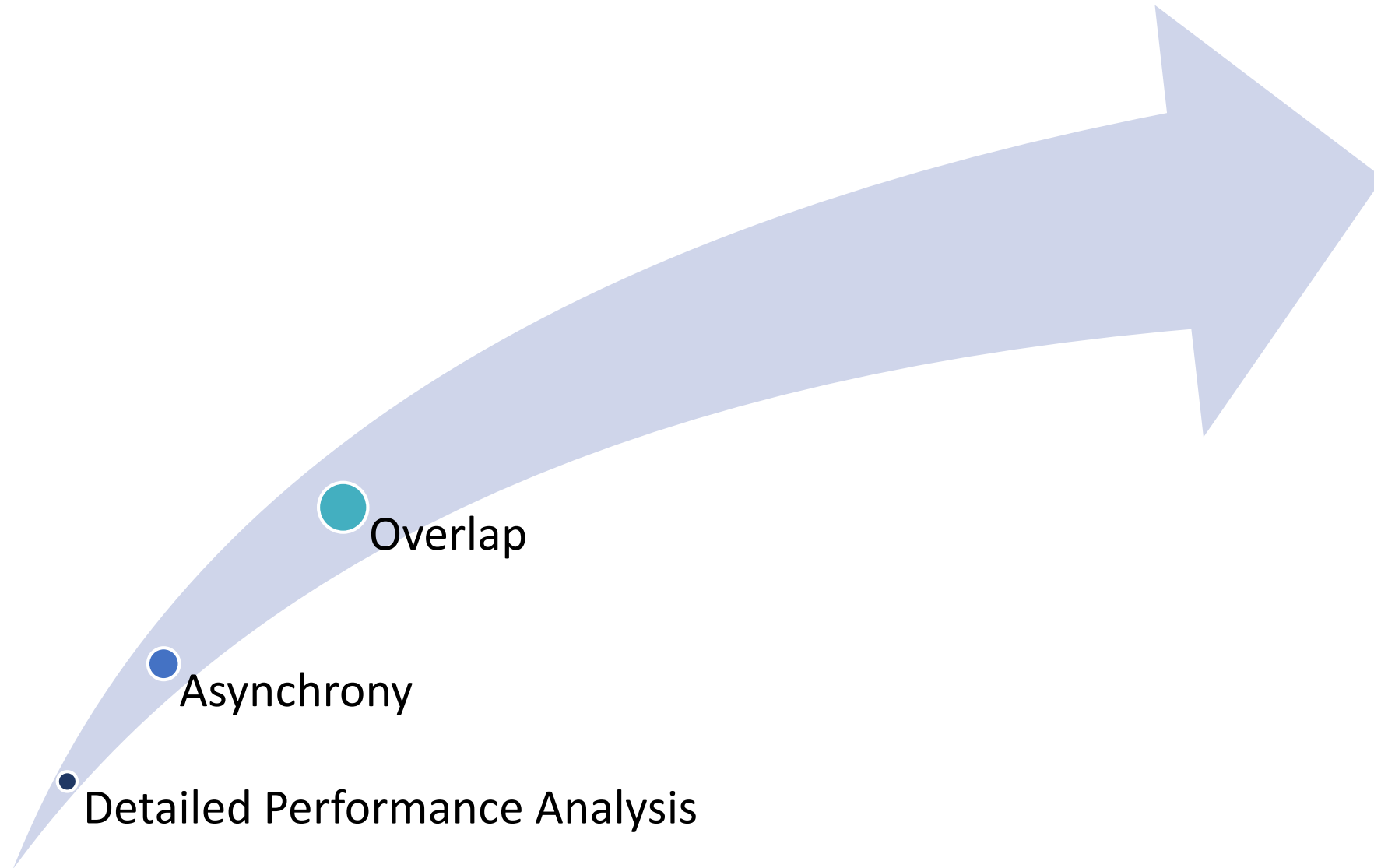
Step by step...



2021/07/07



Step by step...



2021/07/07

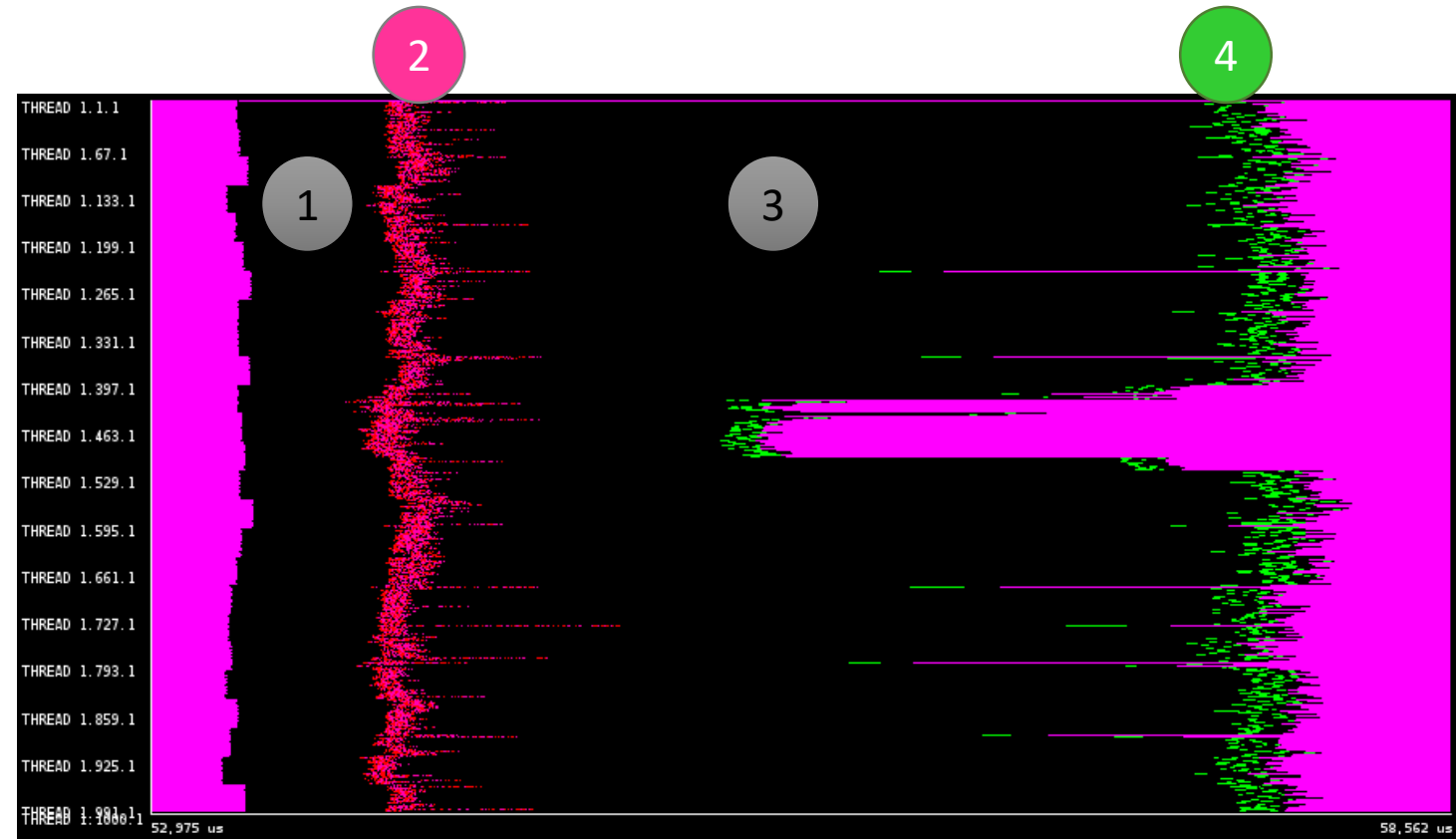


Overlap



- Compute something that does not depend on the communication between the communication and the wait.

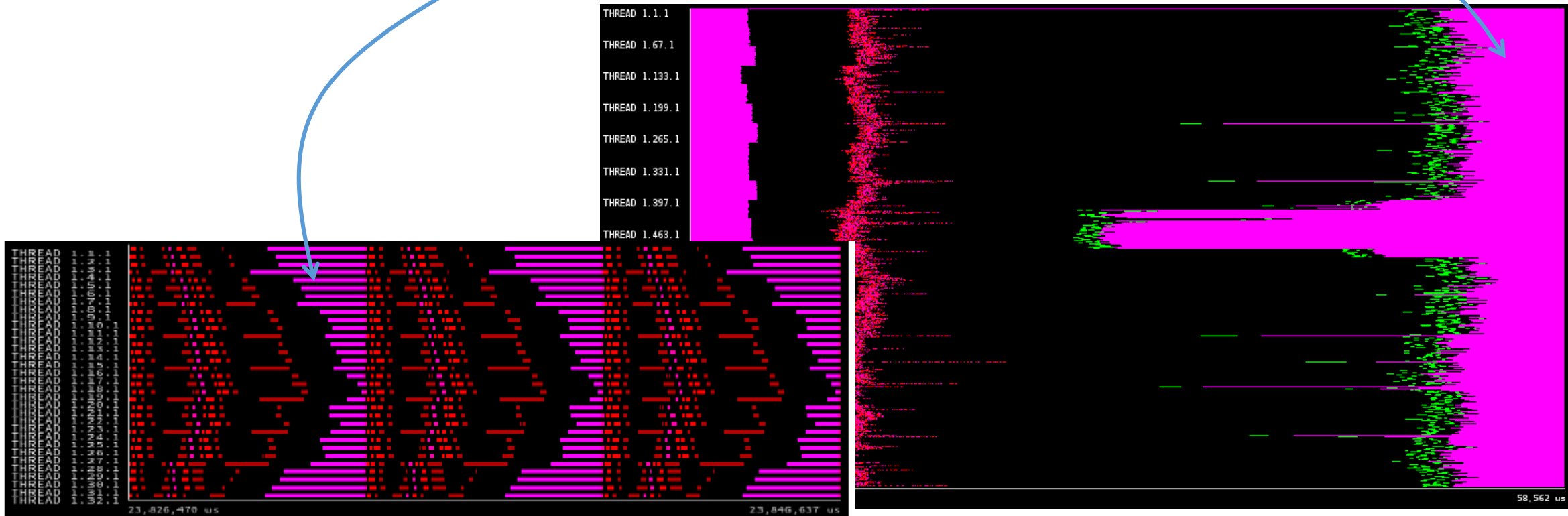
```
for (i=0; i<N; i++){  
    compute_my_boundary() ①  
    for(all my neighbors){  
        lsend(my_boundary) ②  
        lrecv (her_boundary)  
    }  
    compute_my_internal() ③  
    wait_all ④  
}
```



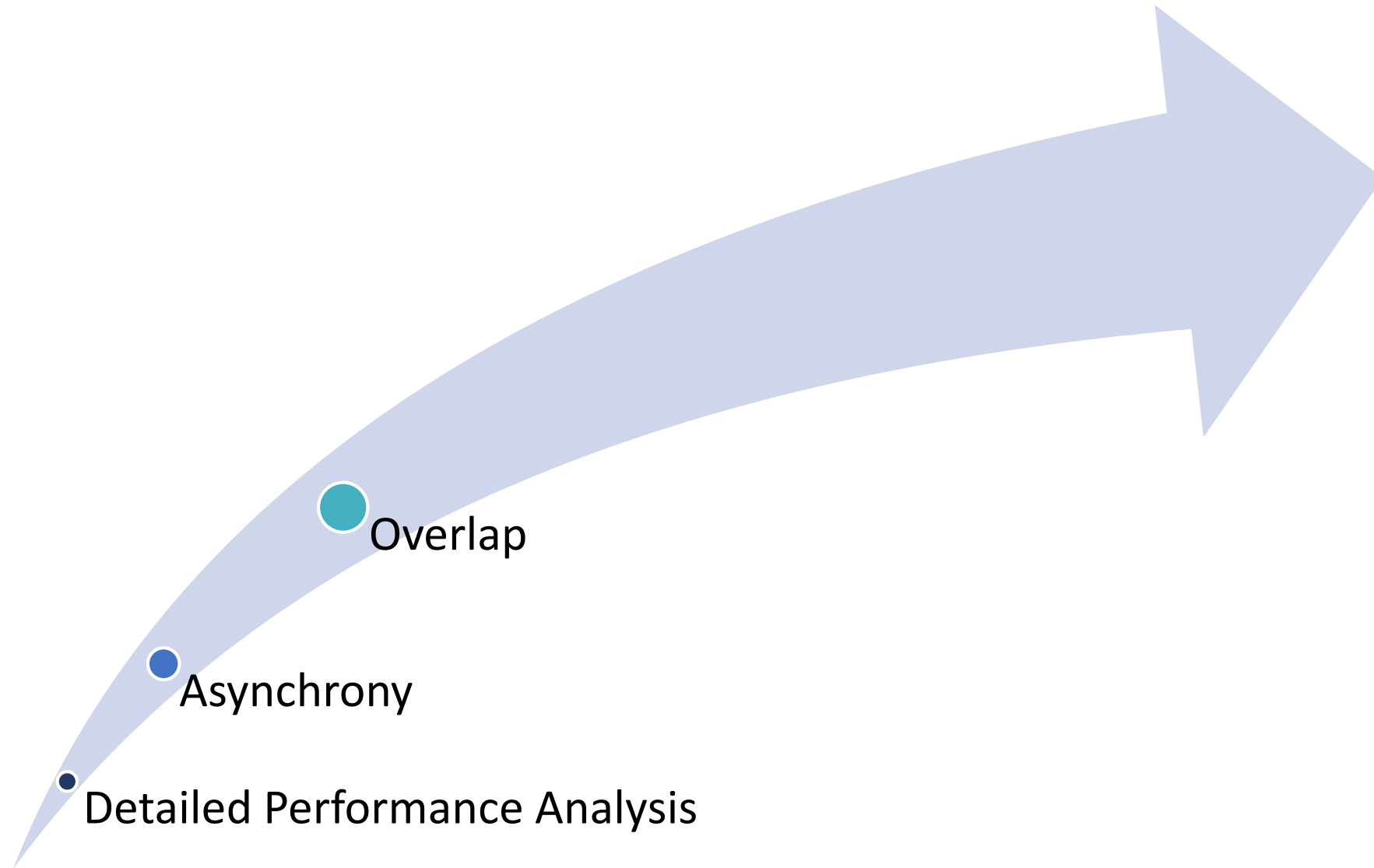
Overlap



- Some applications do it, but not that popular.
- But... load imbalance is still there you “pay” it at the end.



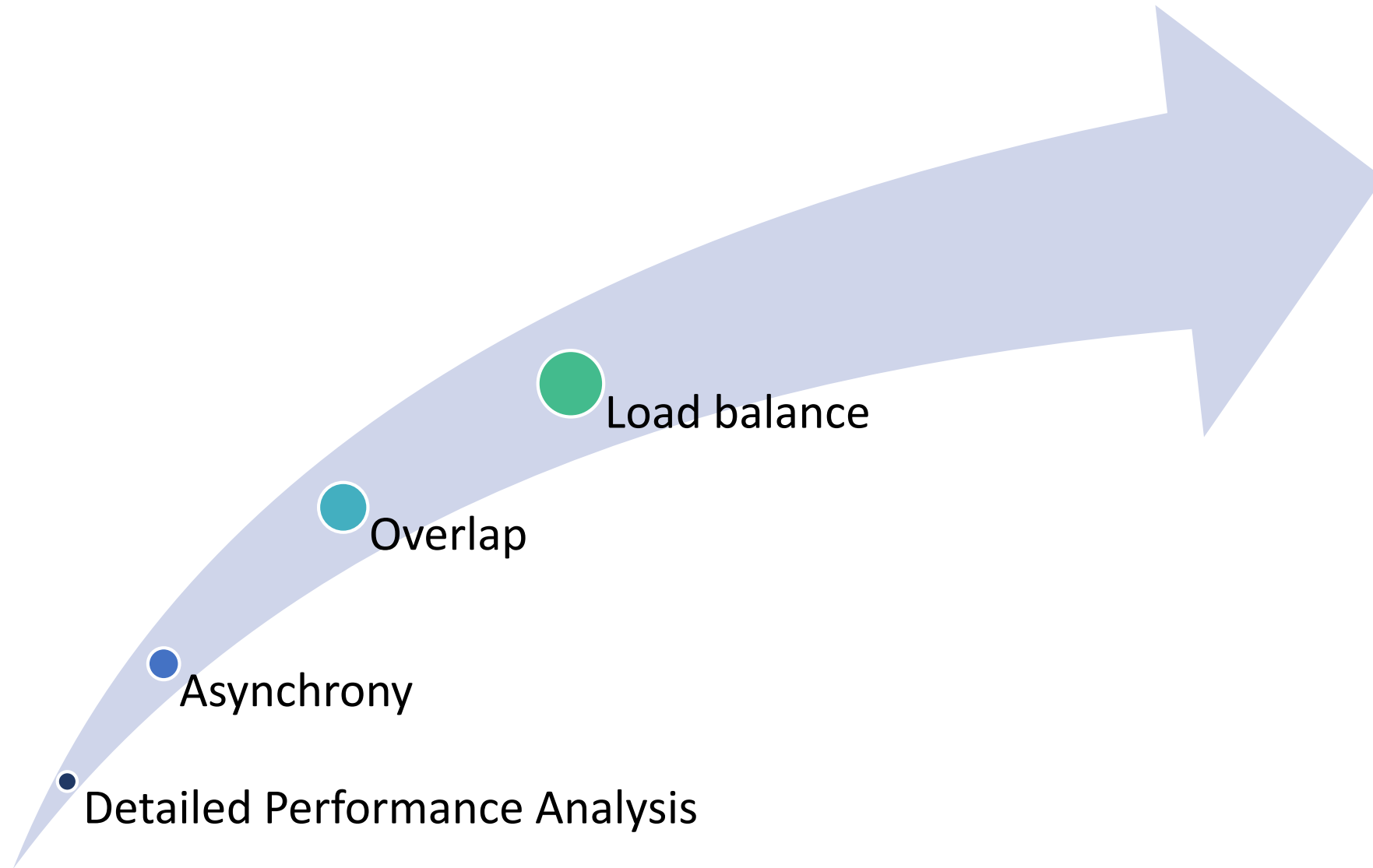
Step by step...



2021/07/07



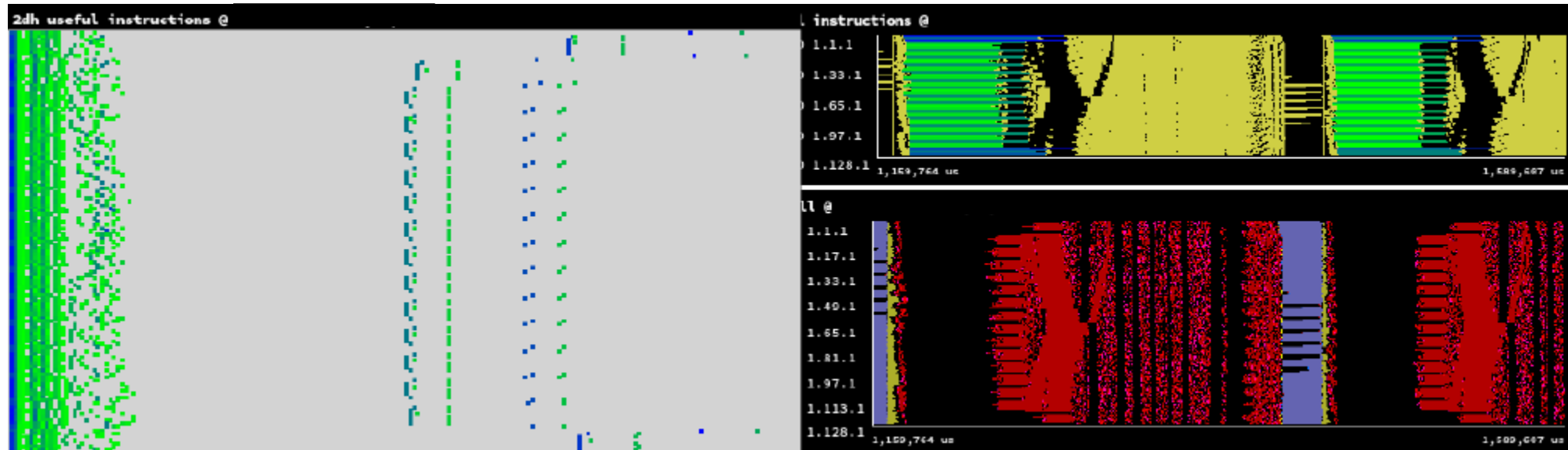
Step by step...



Load imbalance



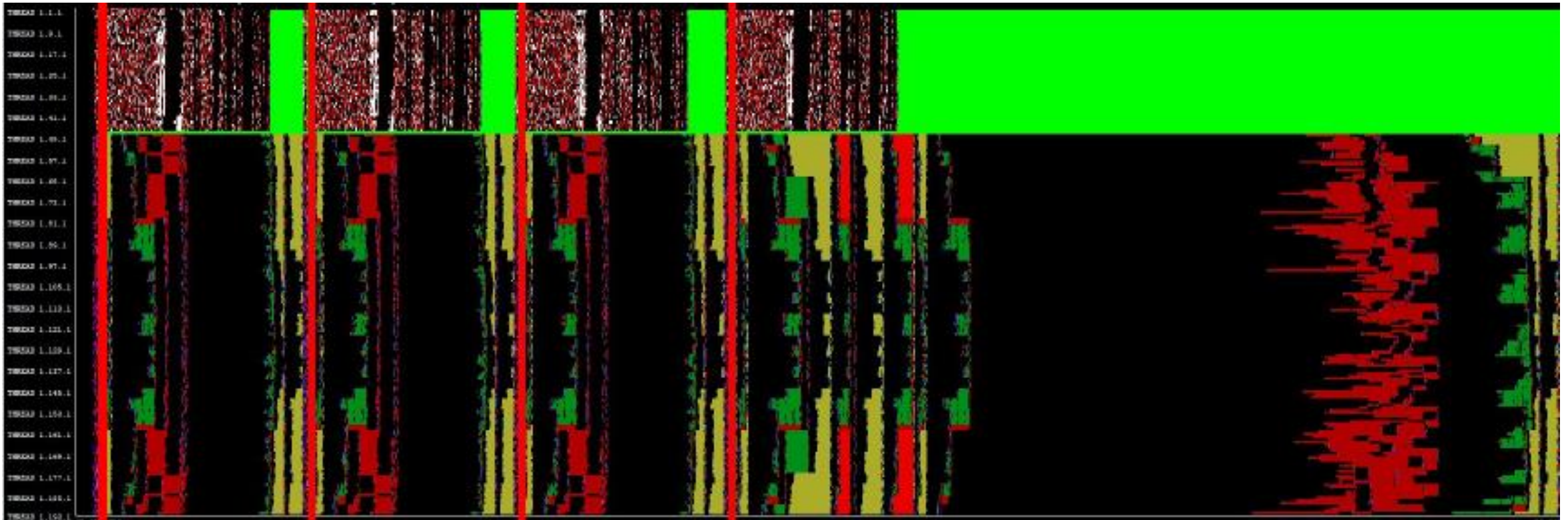
- Where? Why? Source?



Load imbalance @ the problem



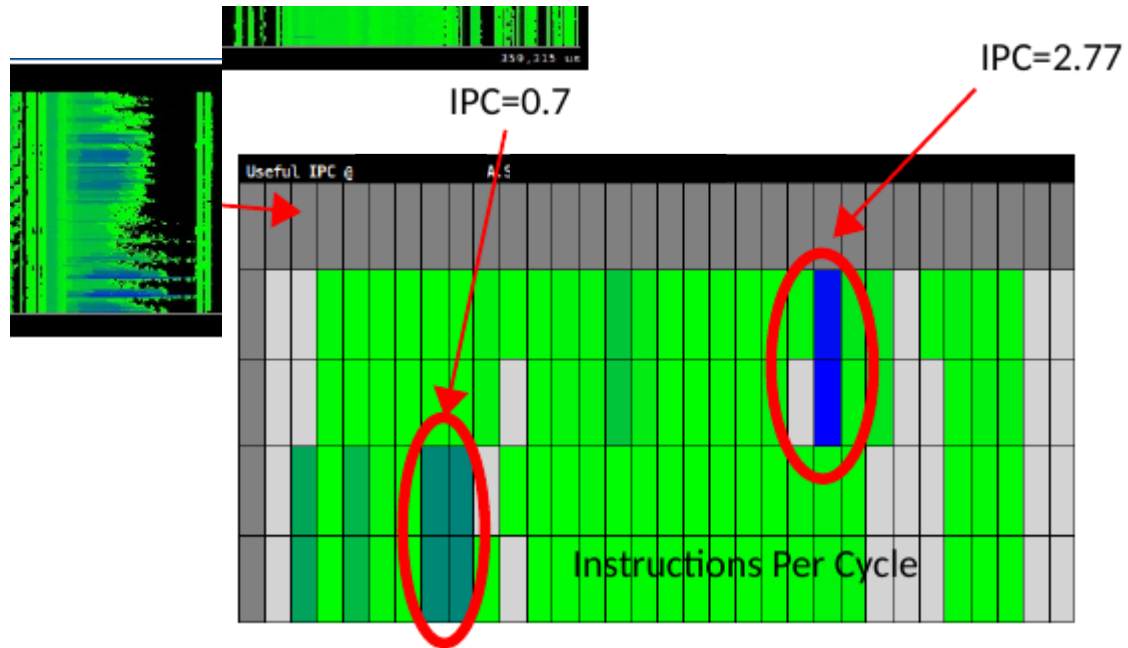
- Coupled problems



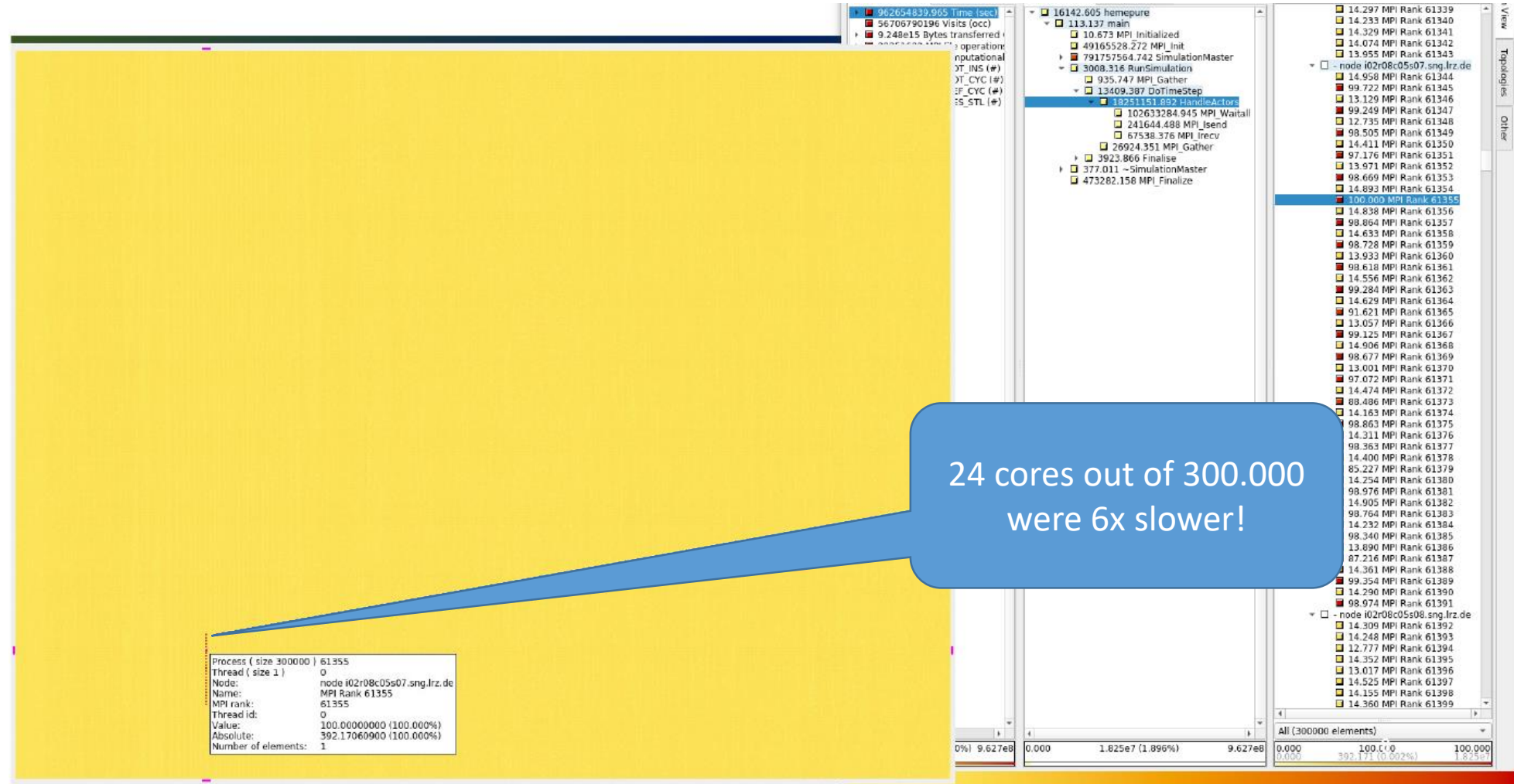
Load imbalance @ IPC



- Same number of instructions
- 30% load imbalance



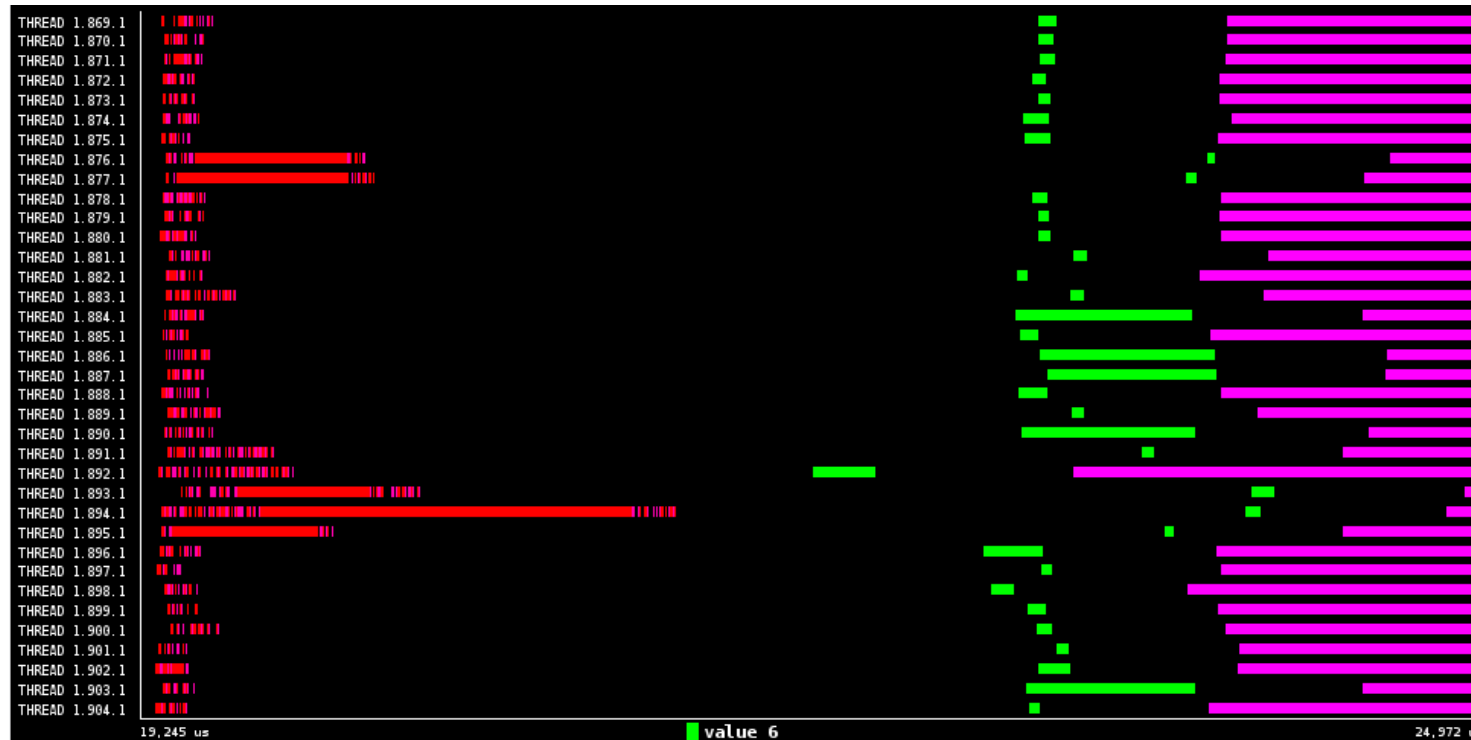
Load imbalance @ hardware



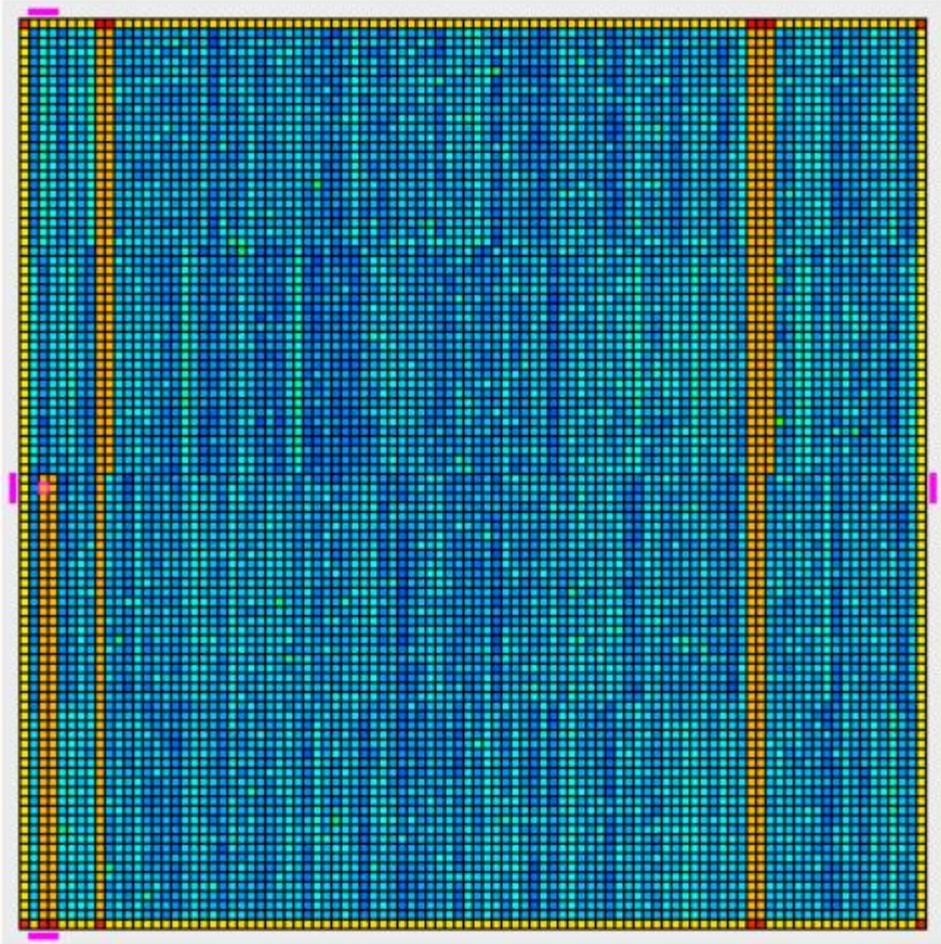
Load imbalance @ system software



- Some Isend calls took longer than usual
- Related to MPI library used, effect disappears when changing MPI library



Load imbalance @ everywhere



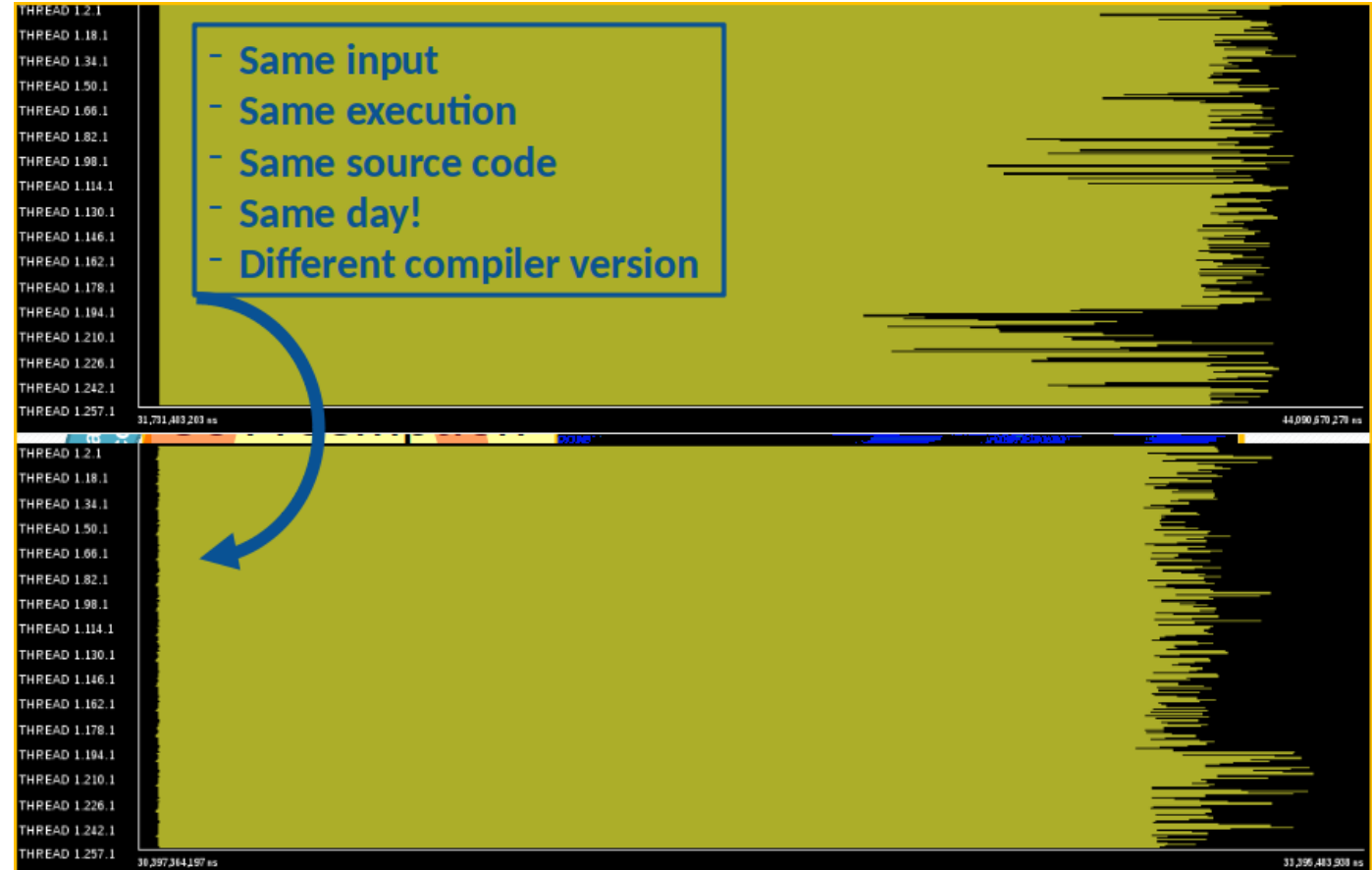
- Hardware issues
- Different performance between “inner” processes and “boundary” processes
- Some code only executed by “boundary” nodes



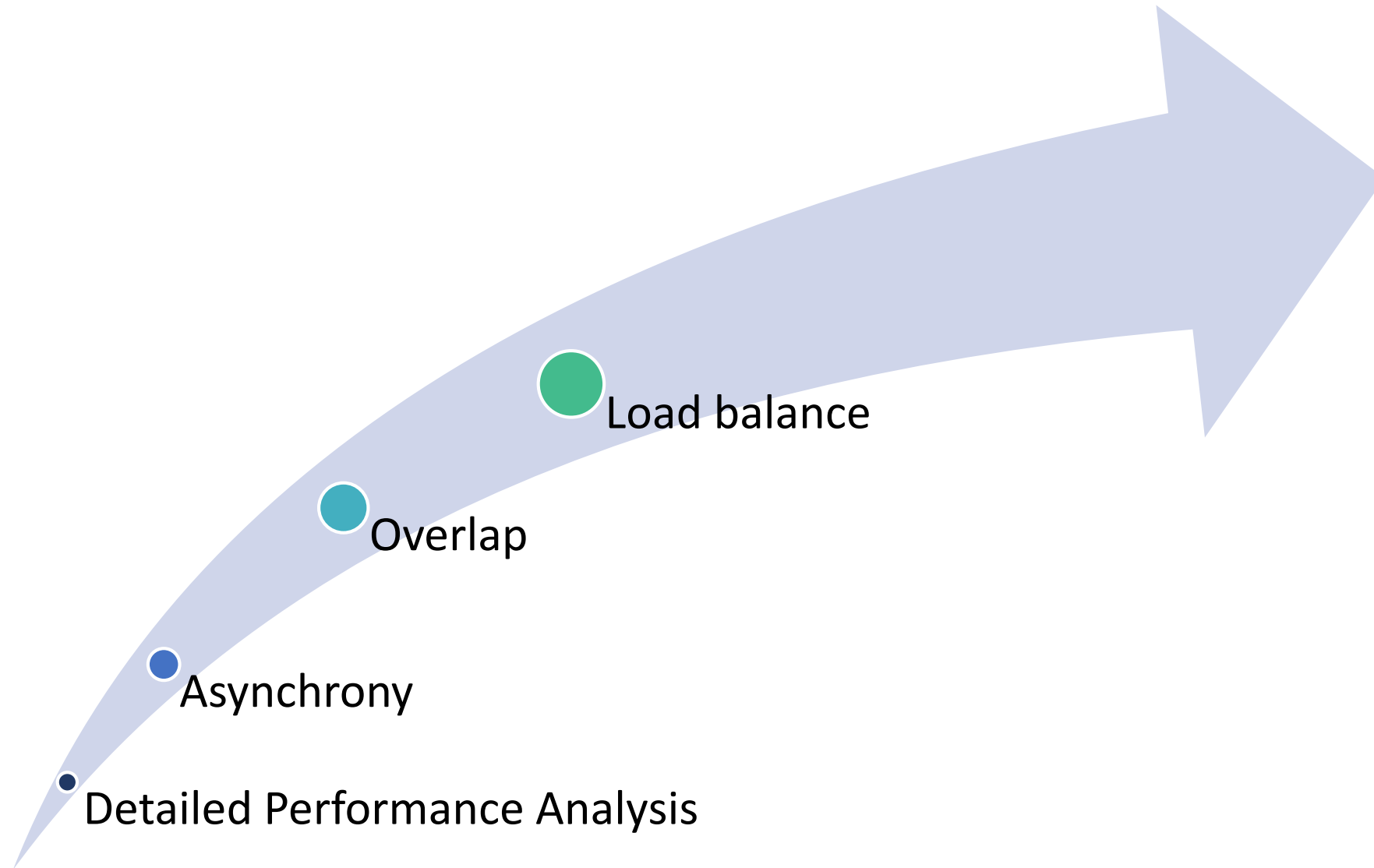
Load imbalance



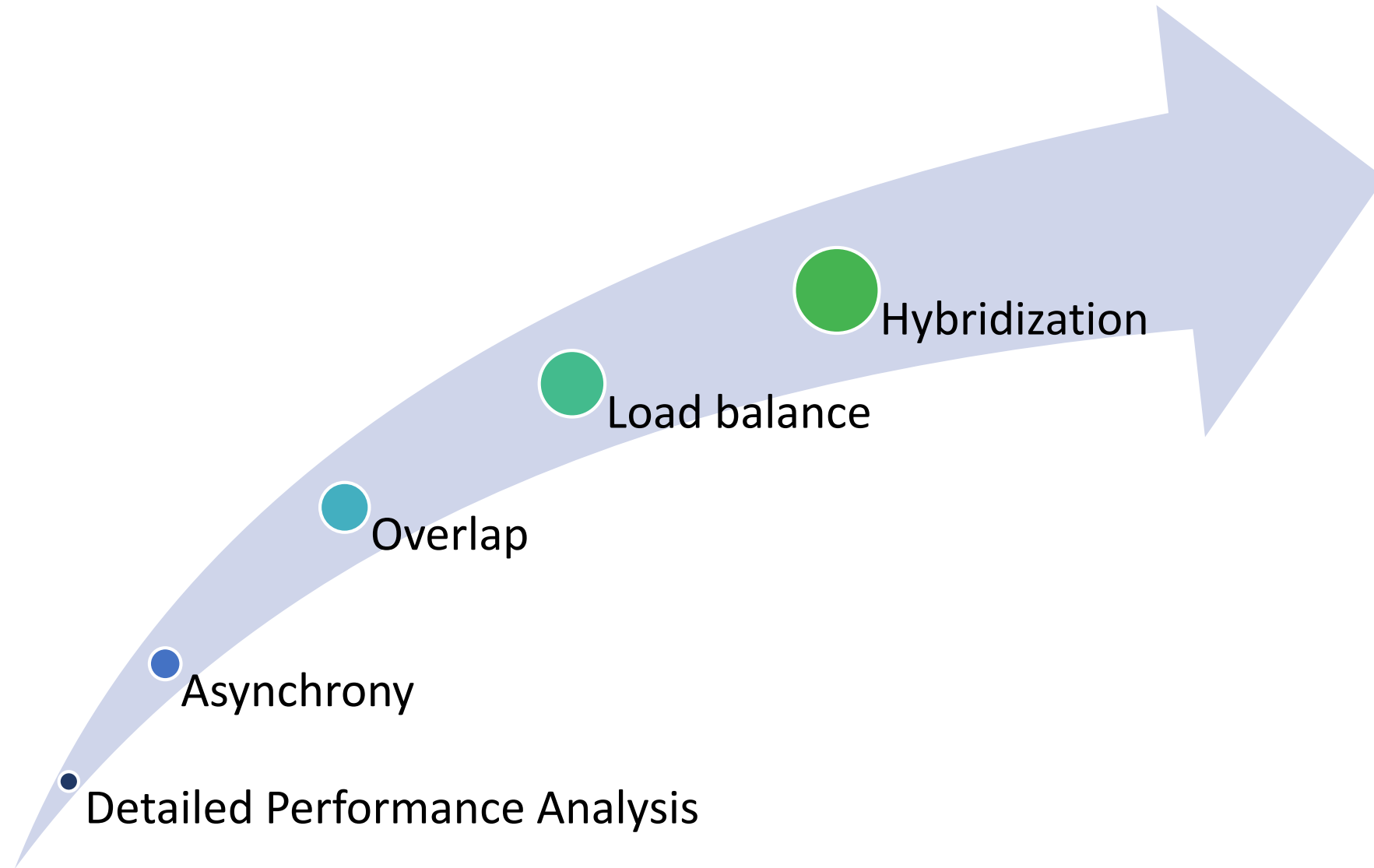
- Cannot be predicted
- Too many factors can produce it
- Do not fight it
 - Adapt!!



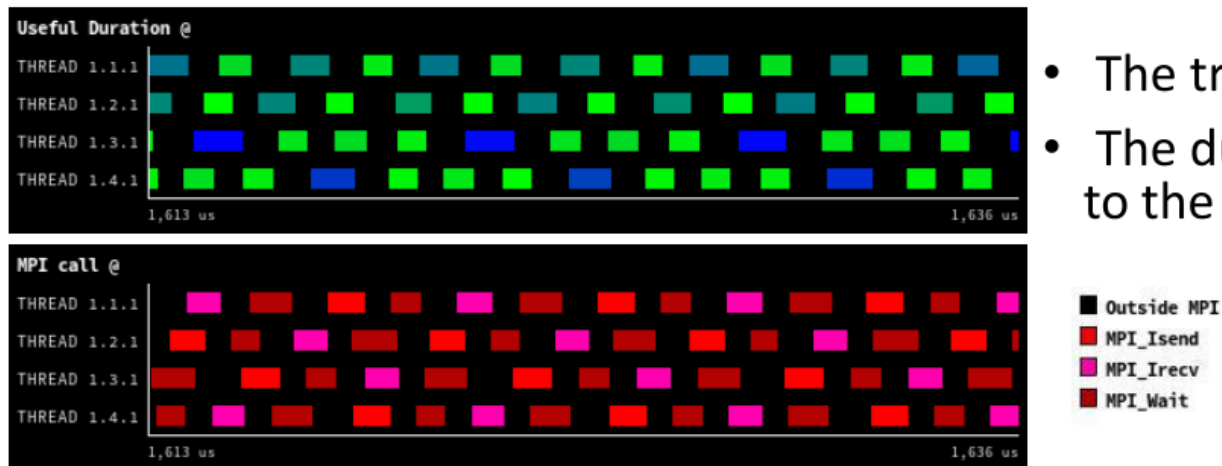
Step by step...



Step by step...



- Why?
 - Because it reduces the amount of communication
 - Because it helps reduce load imbalance
 - Because it helps to overlap communication and computation
 - Because it helps handling low granularities



- The transfer of each call is 16 bytes.
- The duration of the computations is similar to the duration of MPI_Isend and MPI_Irecv!

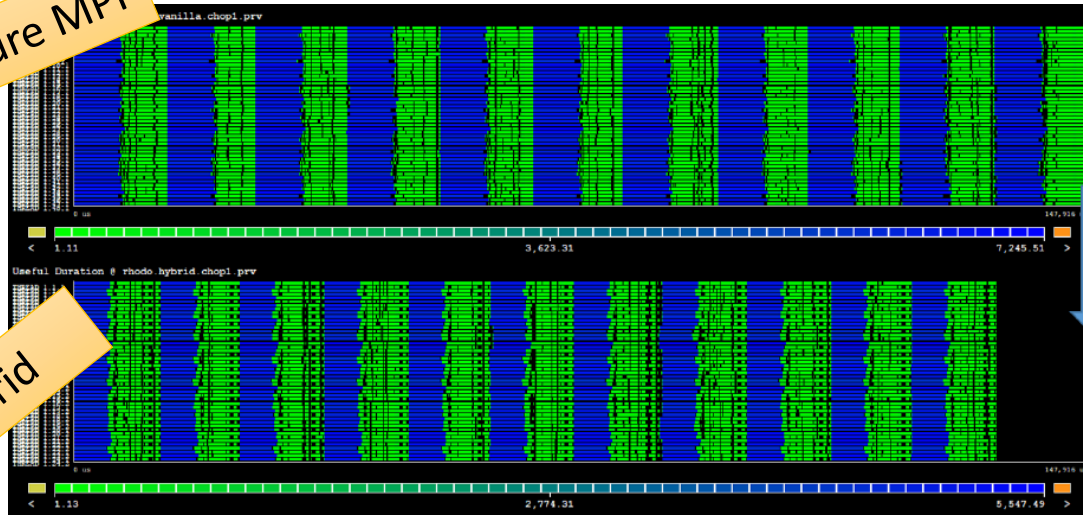
Go Hybrid



Pure MPI

- Effect of hybridizing vs. pure MPI

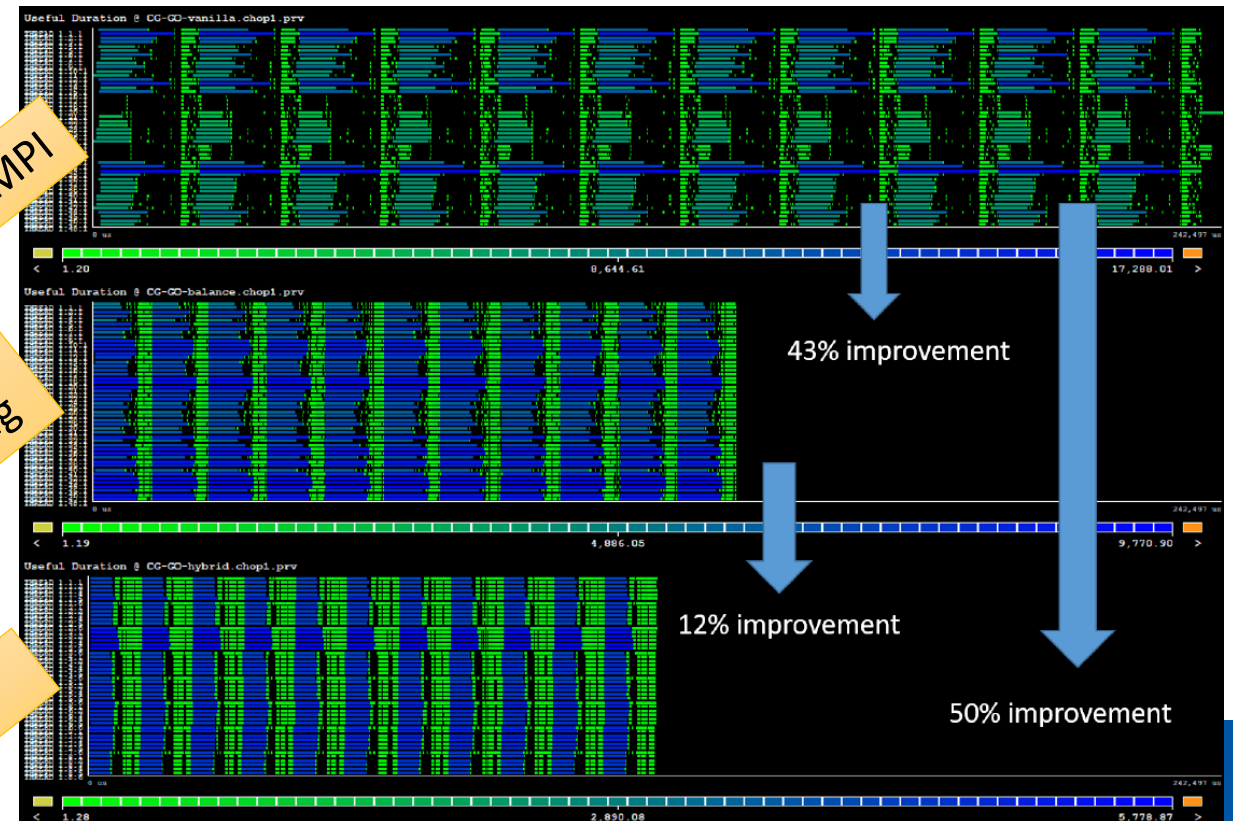
Hybrid



Pure MPI

Hand-coded
load balancing

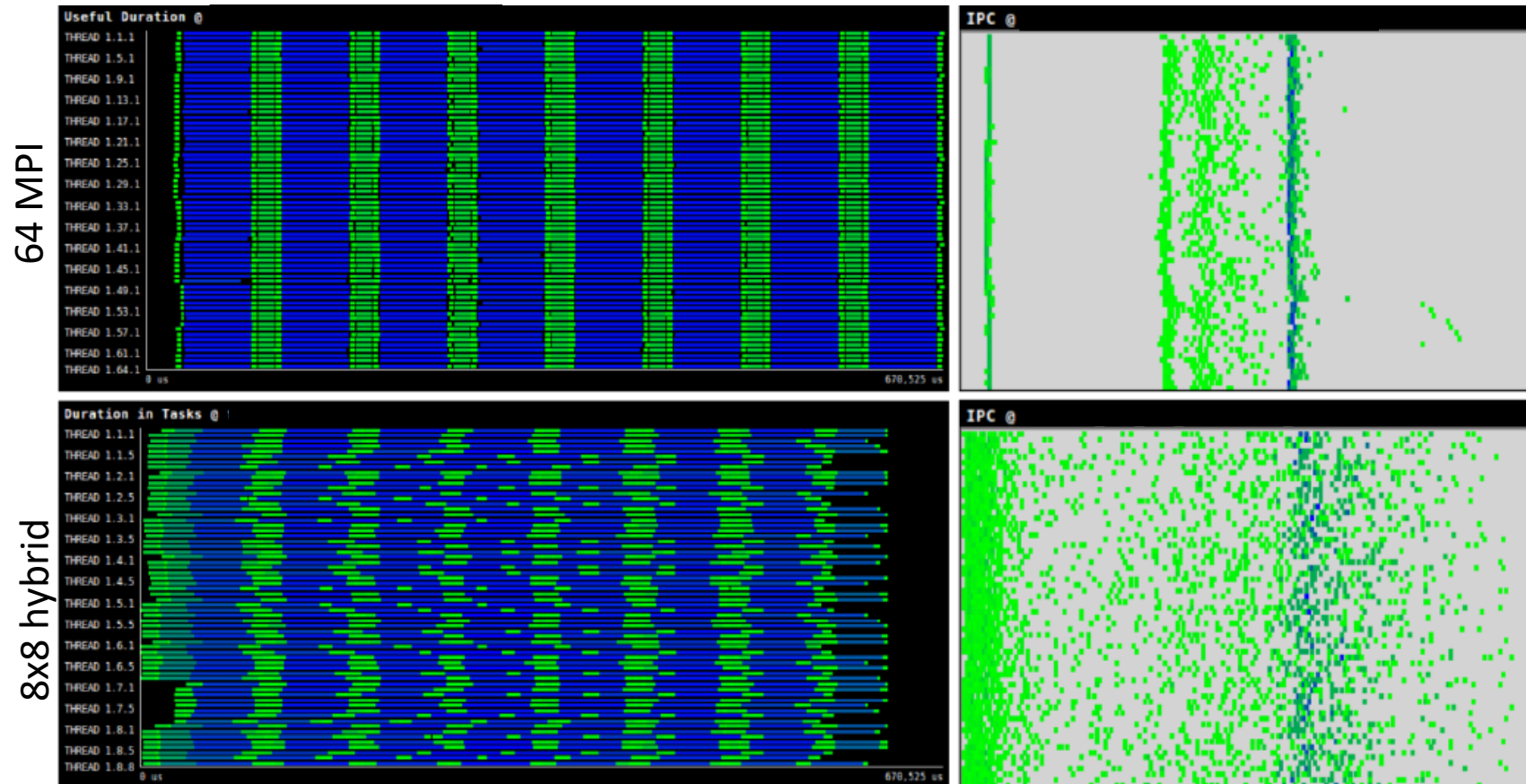
Hybrid



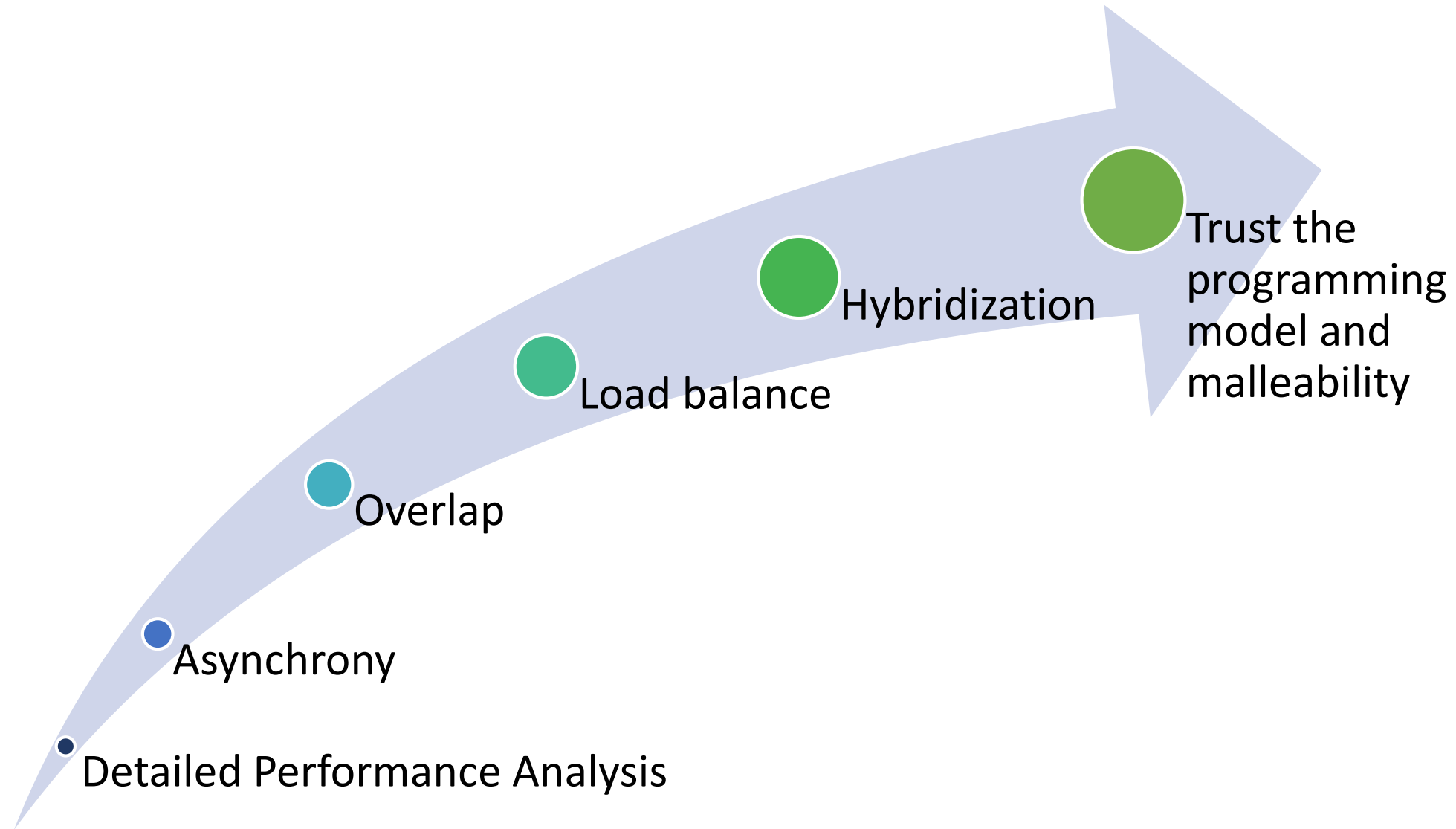
Go Hybrid



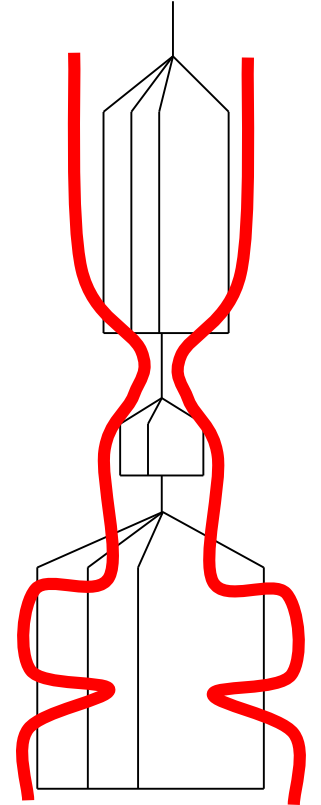
- Helps overlap computation and communication
- Can have good side effects



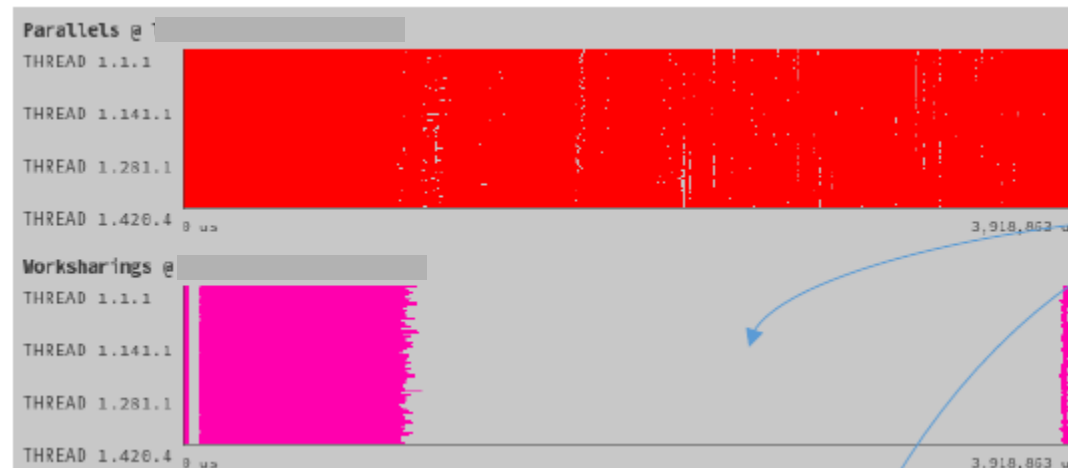
Step by step...



- Property of being adaptable to a change in the environment (resources)
- Where?
 - Programming model
 - Application
- Recommendations:
 - Avoid `omp_get_thread_num`, `Threadprivate`
 - Scheduling → trust the runtime
 - Focus on logic



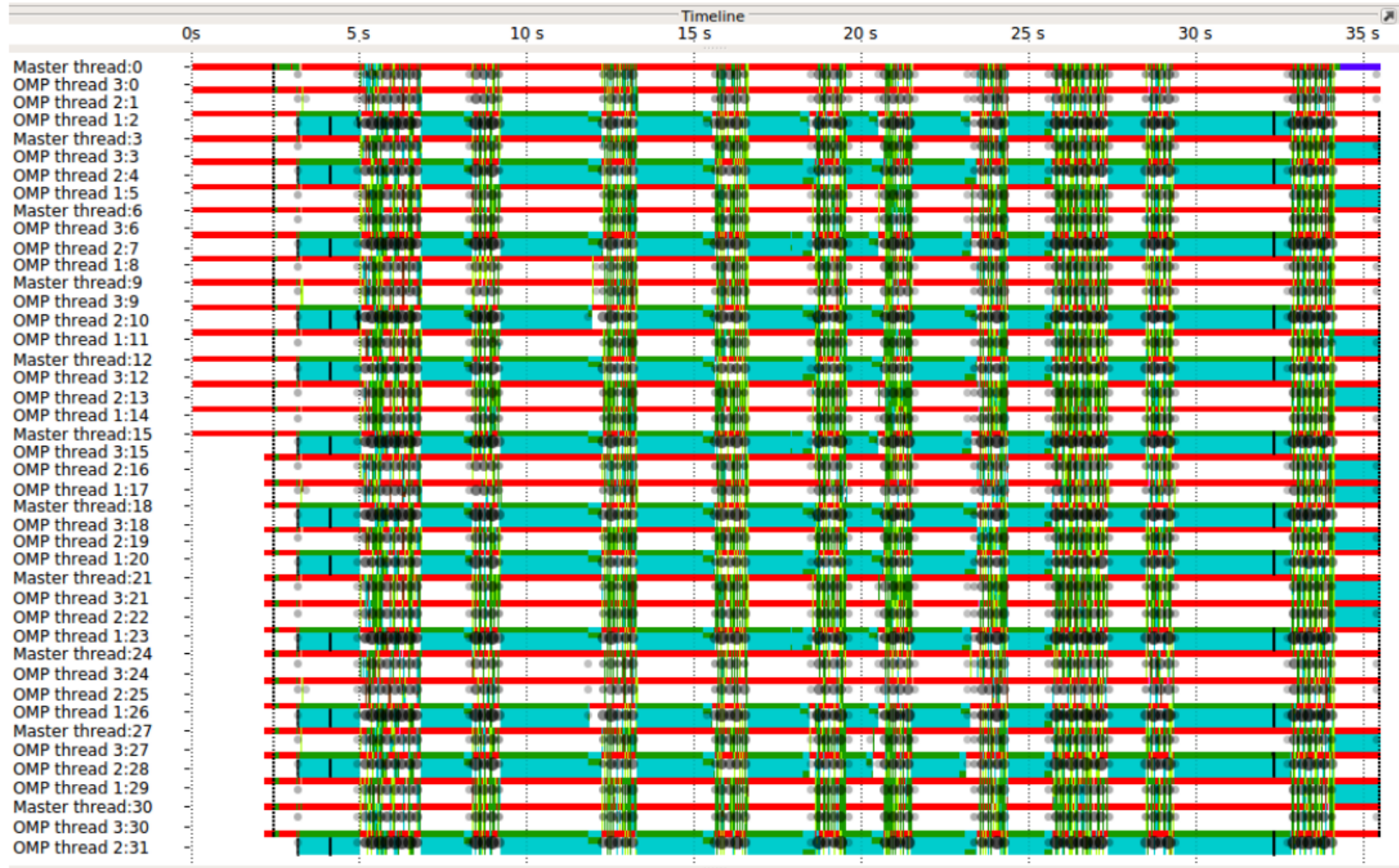
Trust the programming model



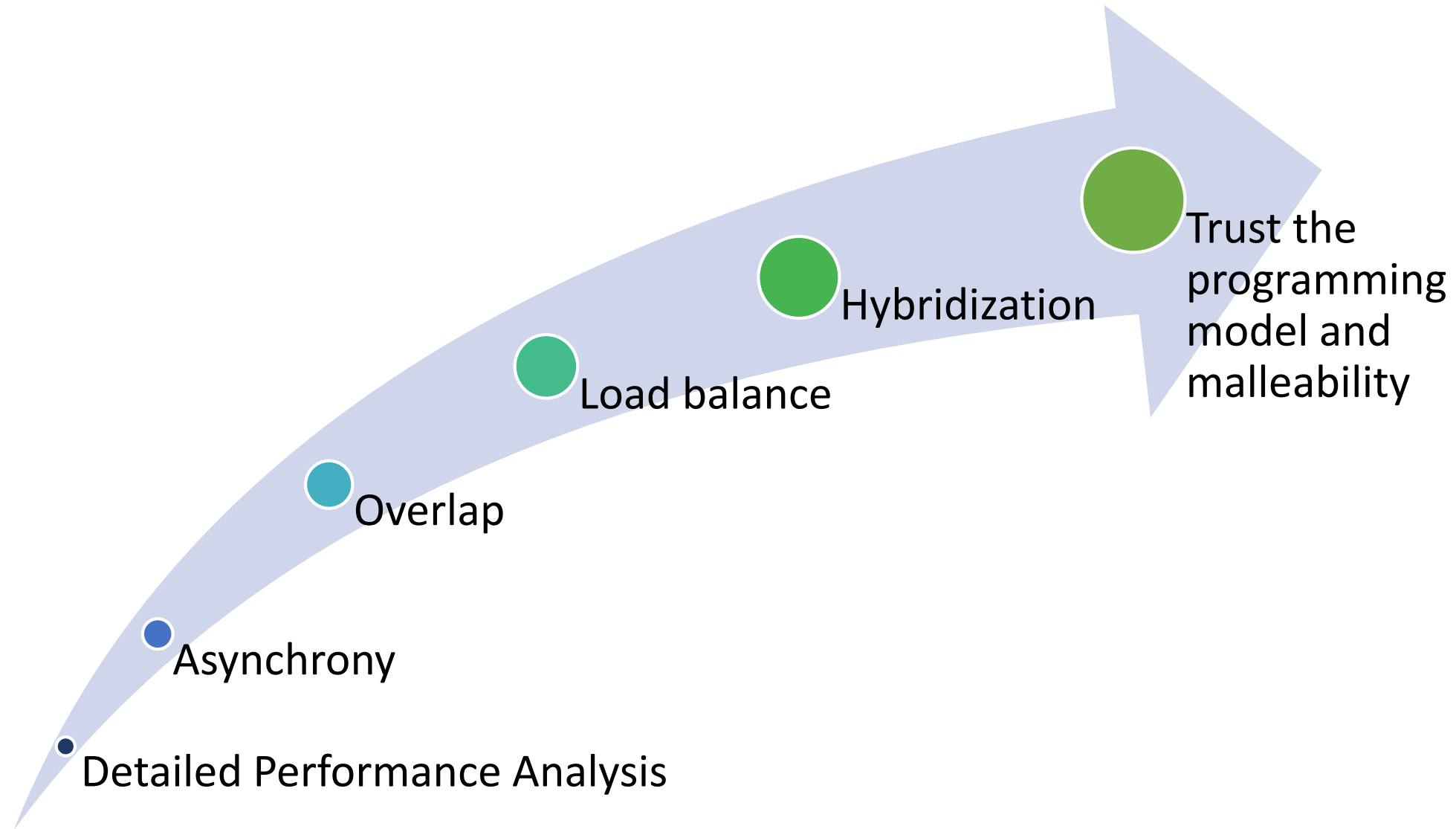
Large fraction of code with parallels without workshares



Trust the programing model



Step by step...



- Do not fly blind
 - Let performance analysis guide you
- Asynchrony is not enough, overlap
- Overlap can hide Communication overhead but not load balance
- Do not fight variability, adapt!
 - Be flexible and malleable
- Trust the programming model
 - Forget about hardware





Performance Optimisation and Productivity

A Centre of Excellence in HPC

Contact:

<https://www.pop-coe.eu>

<mailto:pop@bsc.es>

 @POP_HPC

