The Load Balancing Sandbox

Vincent Reverdy (vreverdy@illinois.edu)

Department of Astronomy
University of Illinois at Urbana-Champaign

Laboratory Universe and Theories
Paris Observatory

C++ Standards Committee
ISO

April 16th, 2019

Summary

Summary

Summary

Proposal for a new project

- Brainstorming started last year at the 8th JLESC meeting with Laércio Lima Pilla
- Collaboration with INRIA and Paris Observatory
- Looking for new collaborators
- Nothing is set yet: new ideas are very welcome!

Summary

Proposal for a new project

- Brainstorming started last year at the 8th JLESC meeting with Laércio Lima Pilla
- Collaboration with INRIA and Paris Observatory
- Looking for new collaborators
- Nothing is set yet: new ideas are very welcome!

Main idea: a load balancing sandbox

Creating a framework to facilitate the development, implementation, deployment and maintenance of load balancing algorithms $\Rightarrow$ Abstraction is key

Summary

Proposal for a new project

- Brainstorming started last year at the 8th JLESC meeting with Laércio Lima Pilla
- Collaboration with INRIA and Paris Observatory
- Looking for new collaborators
- Nothing is set yet: new ideas are very welcome!

Main idea: a load balancing sandbox

Creating a framework to facilitate the development, implementation, deployment and maintenance of load balancing algorithms $\Rightarrow$ Abstraction is key

What is it about?

- Software architecture
- Abstraction
- Standardization
- Advances in programming languages
- Modern C++
- . . . and load balancing

Summary

Proposal for a new project

- Brainstorming started last year at the 8th JLESC meeting with Laércio Lima Pilla
- Collaboration with INRIA and Paris Observatory
- Looking for new collaborators
- Nothing is set yet: new ideas are very welcome!

Main idea: a load balancing sandbox

Creating a framework to facilitate the development, implementation, deployment and maintenance of load balancing algorithms $\Rightarrow$ Abstraction is key

What is it about?

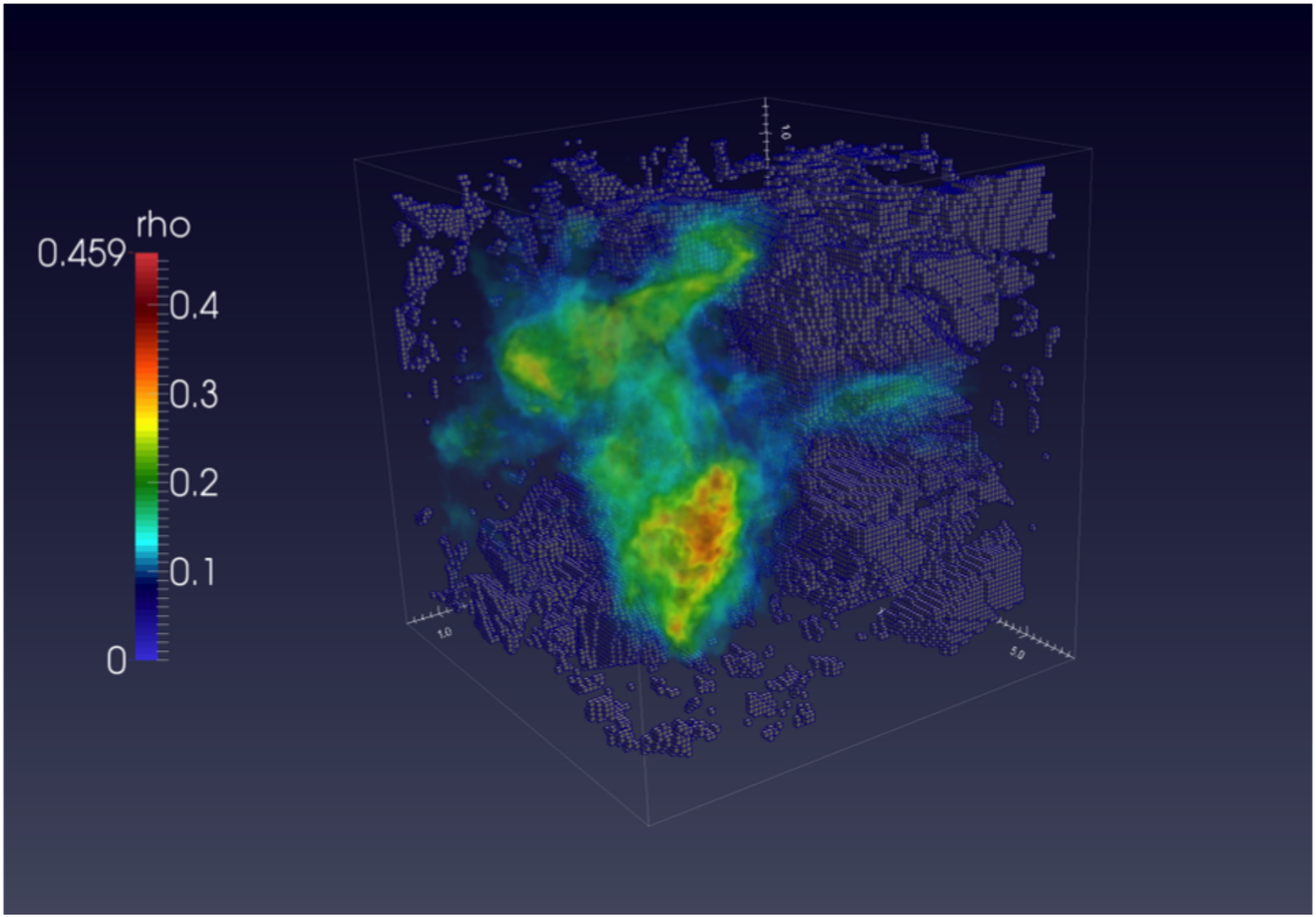
- Software architecture
- Abstraction
- Standardization
- Advances in programming languages
- Modern C++
- ...and load balancing

What is it not about?

- A particular application domain
- Building a runtime system
- Load balancing theory
- Specific load balancing algorithms

Problem

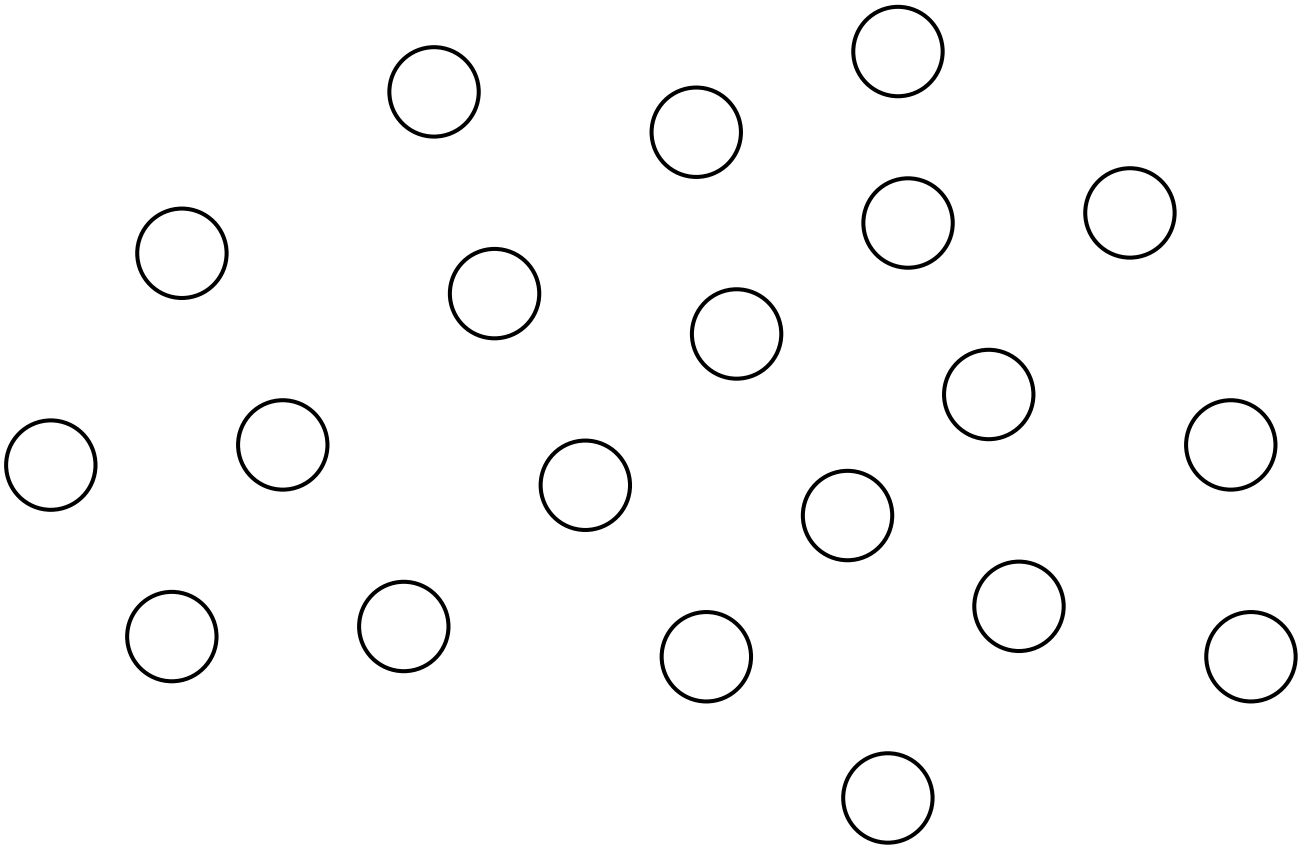
The problem



The problem

Components

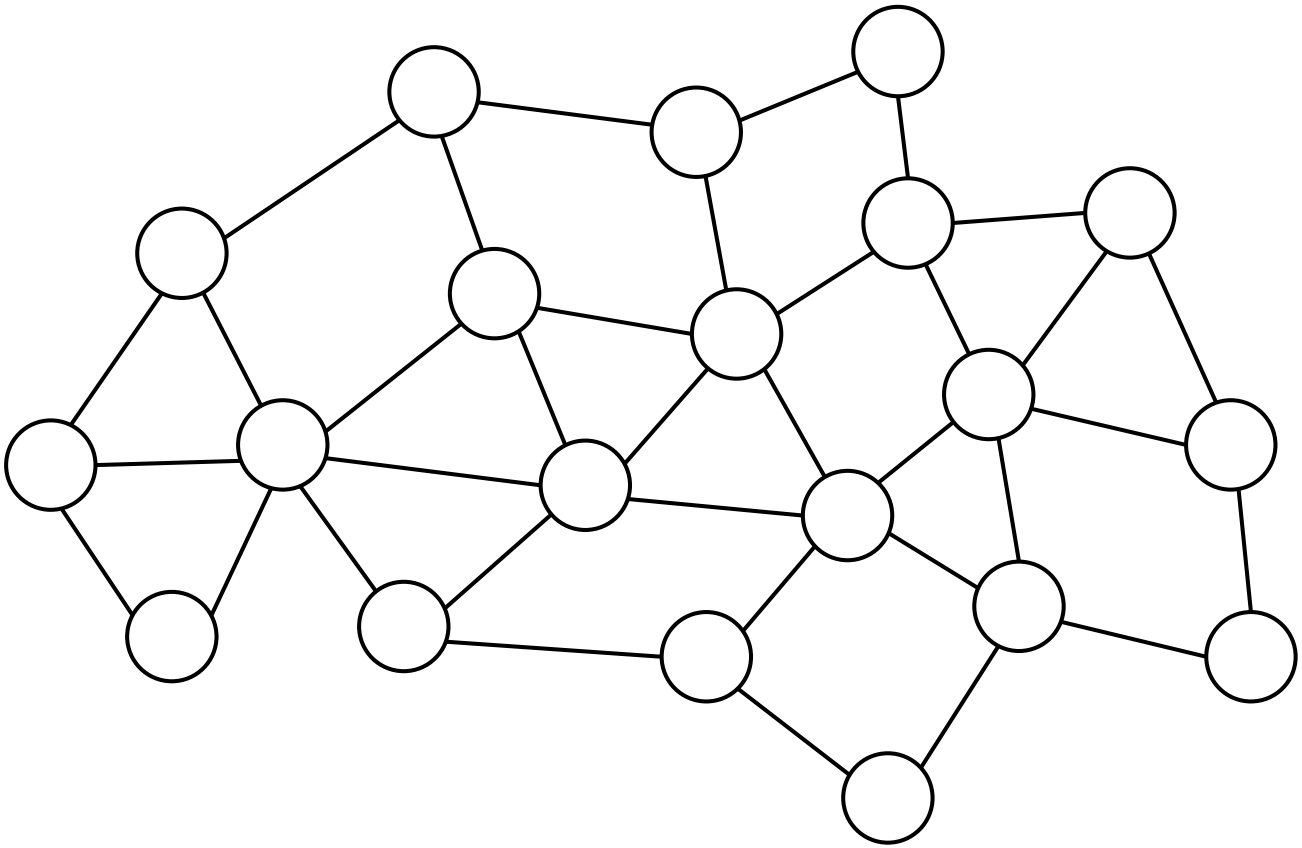
N tasks/coroutines



The problem

Components

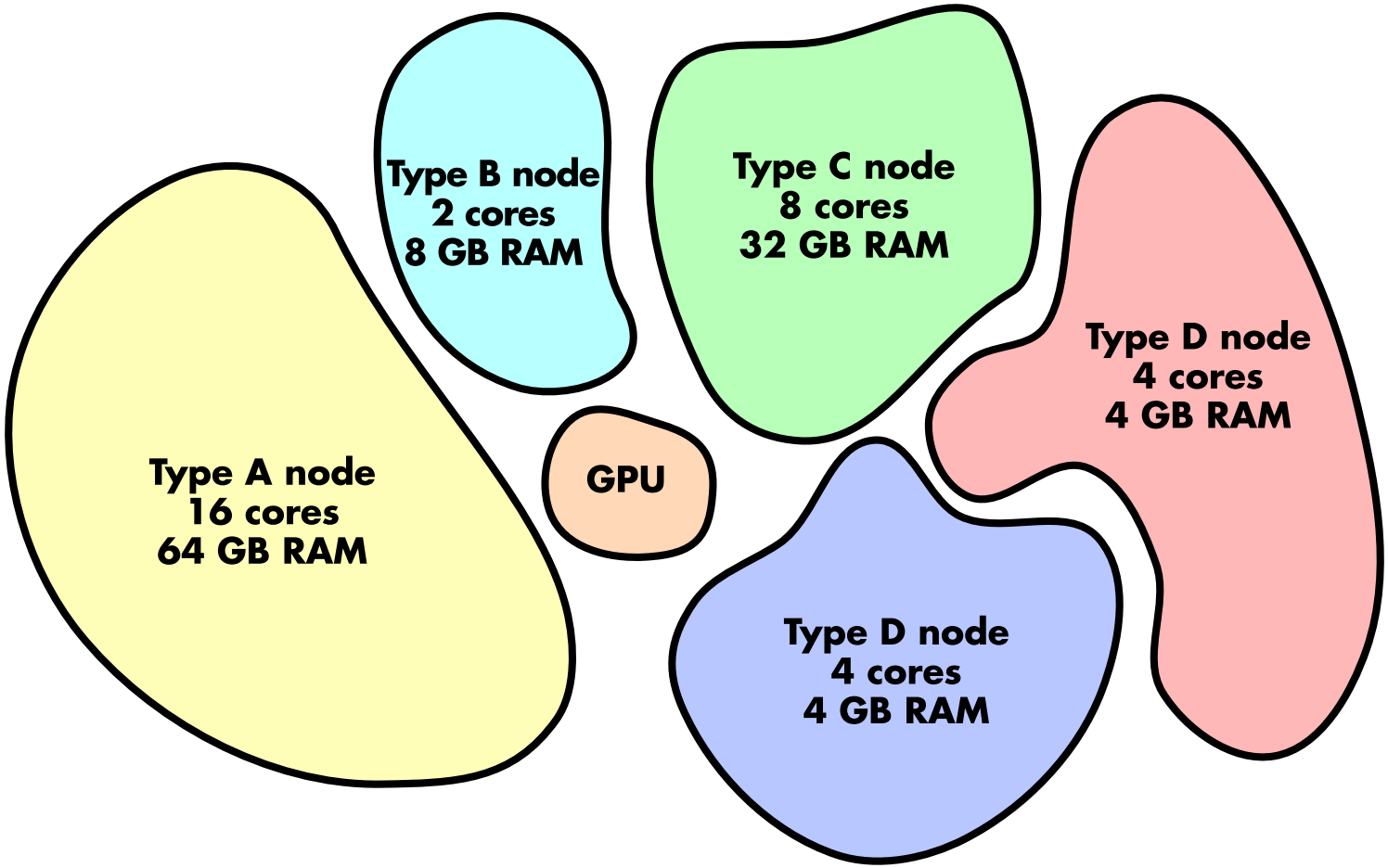
N tasks/coroutines + dependency graph



The problem

Components

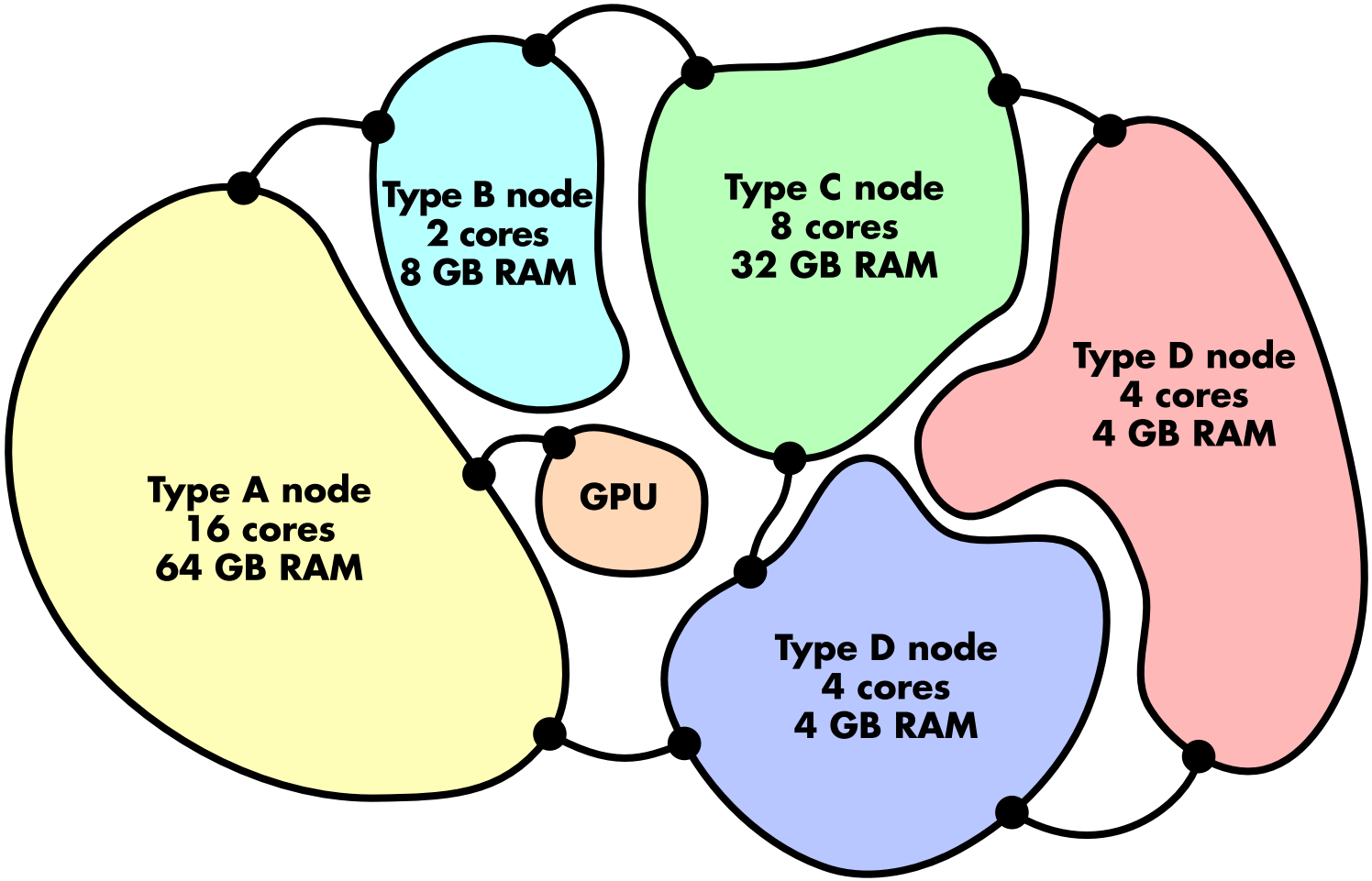
N tasks/coroutines + dependency graph + M distributed nodes



The problem

Components

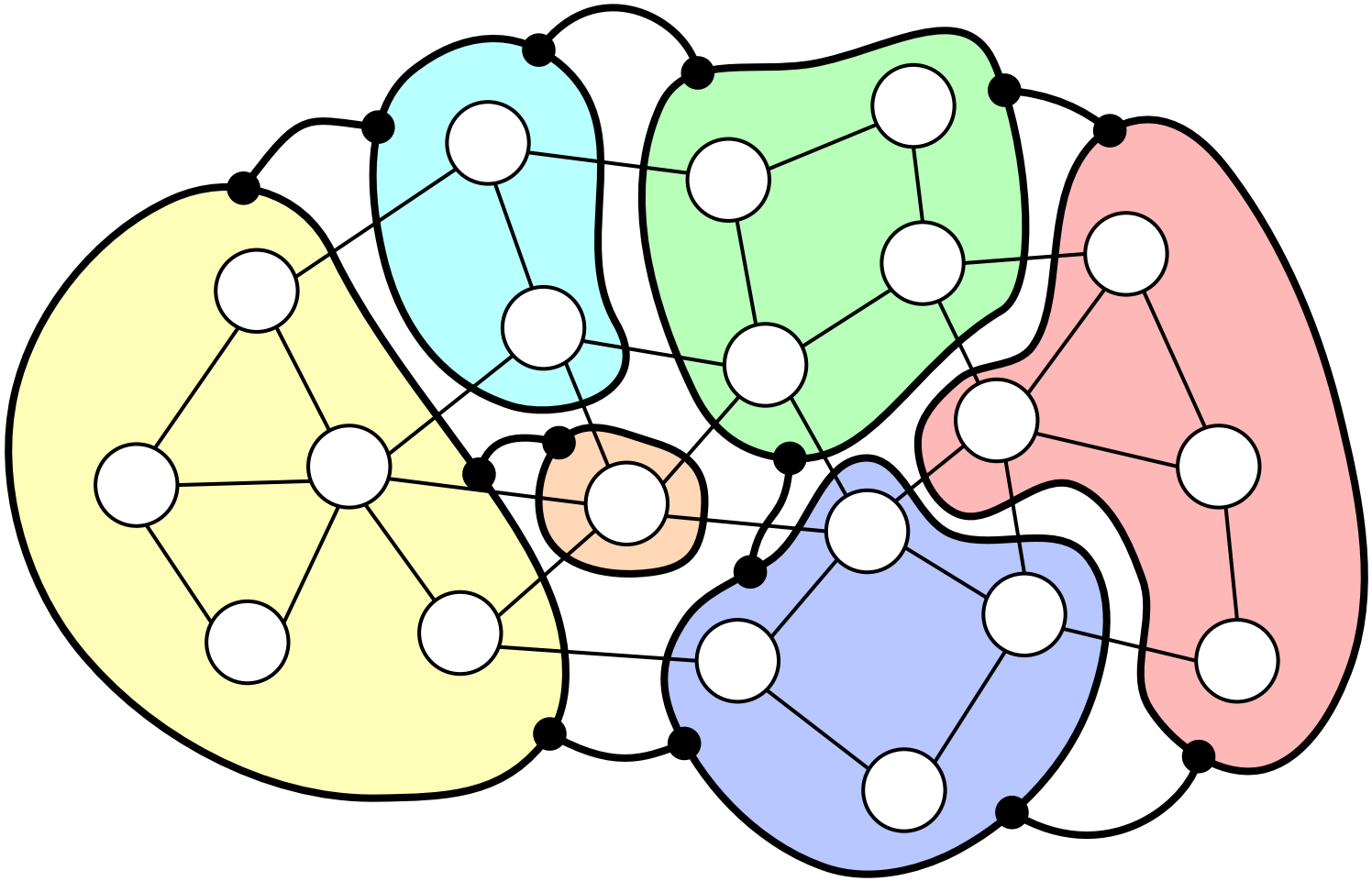
N tasks/coroutines + dependency graph + M distributed nodes + hardware topology



The problem

Components

N tasks/coroutines + dependency graph + M distributed nodes + hardware topology



The problem

Problem summary

Load balancing: easy to describe on a whiteboard, a nightmare to implement.

Suggested solution

What if we had a common framework that would abstract hardware, runtime systems and softwares, and allow rapid prototyping and deployment of load balancing algorithms?

Goals

- Speeding up: Theory $\Rightarrow$ Implementation
- Speeding up: Experimentation $\Rightarrow$ Production
- Facilitating: Interoperability

Approaches

Original approach

- Software architecture, abstraction and standardization
- Implementation, languages and type systems
- Emulation, visualization, and prototyping tool
- Interface the load balancing framework with AI tools

Approaches

Original approach

- Software architecture, abstraction and standardization
- Implementation, languages and type systems
- Emulation, visualization, and prototyping tool
- Interface the load balancing framework with AI tools

After 8th JLESC meeting

- Abstraction of load balacing
- Implementation of the resulting abstractions
- Simulation of applications
- Application to numerical astrophysics

Approaches

Original approach

- Software architecture, abstraction and standardization
- Implementation, languages and type systems
- Emulation, visualization, and prototyping tool
- Interface the load balancing framework with AI tools

After 8th JLESC meeting

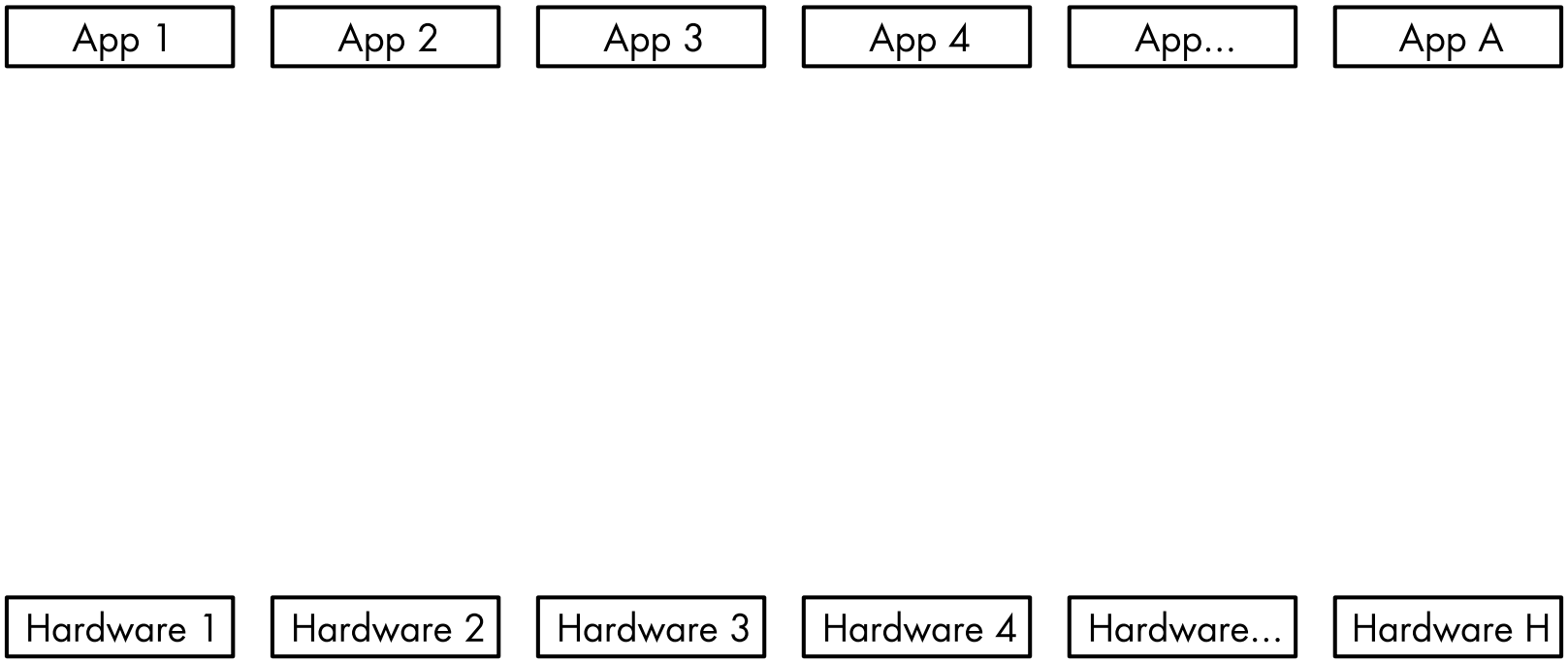
- Abstraction of load balacing
- Implementation of the resulting abstractions
- Simulation of applications
- Application to numerical astrophysics

Now: keep it simple

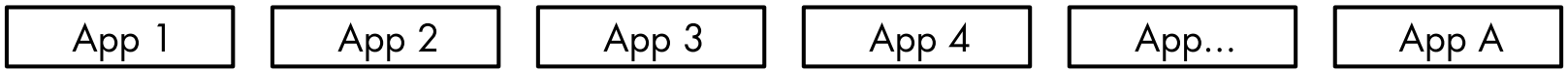
- Standardization of load balancing abstractions

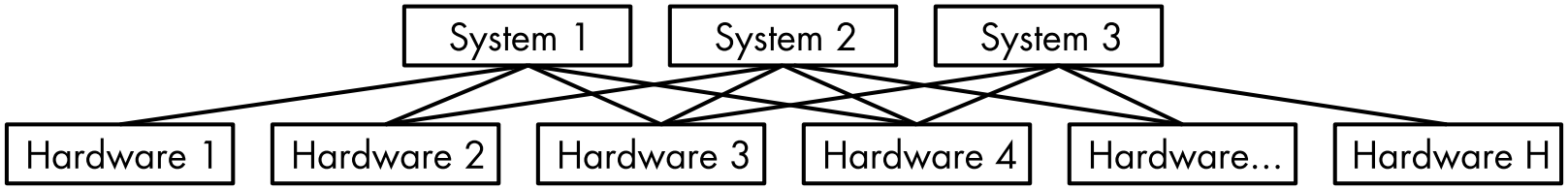
Interfacing

Interfacing

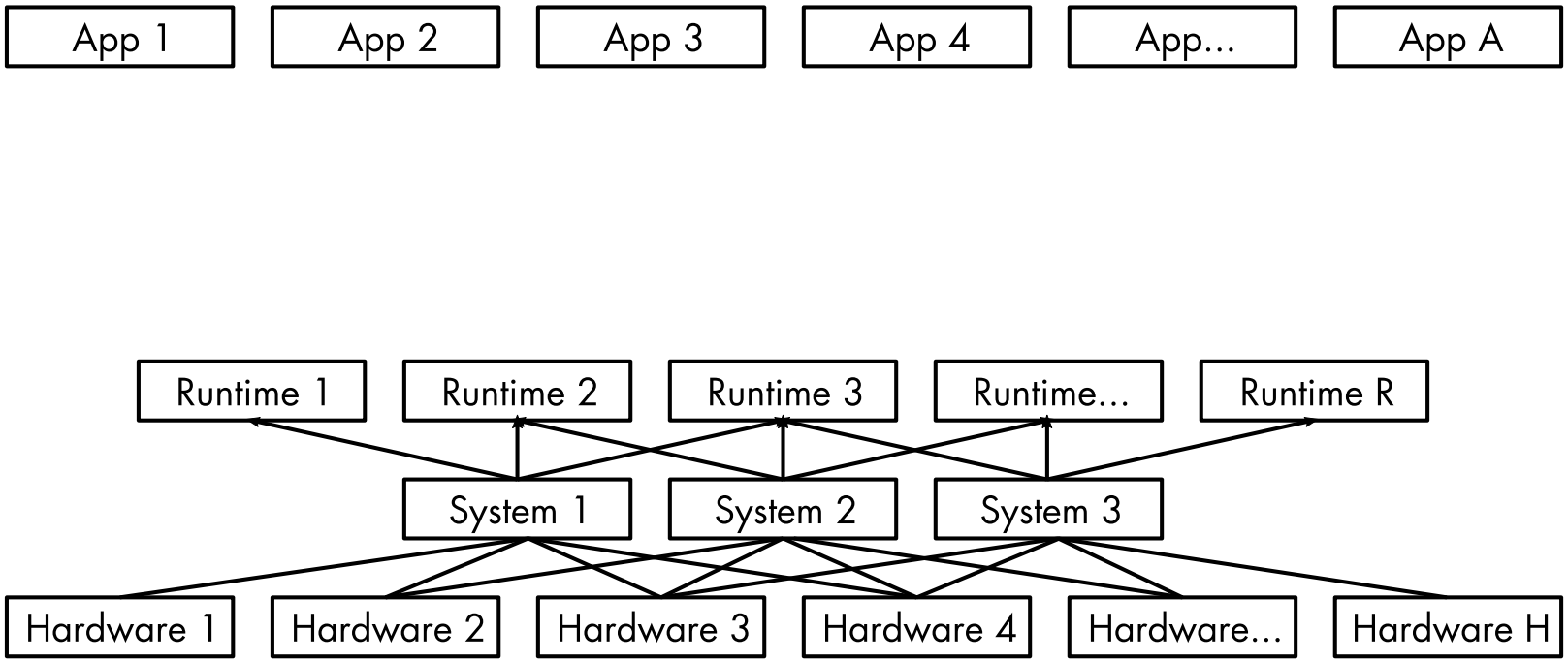


Interfacing

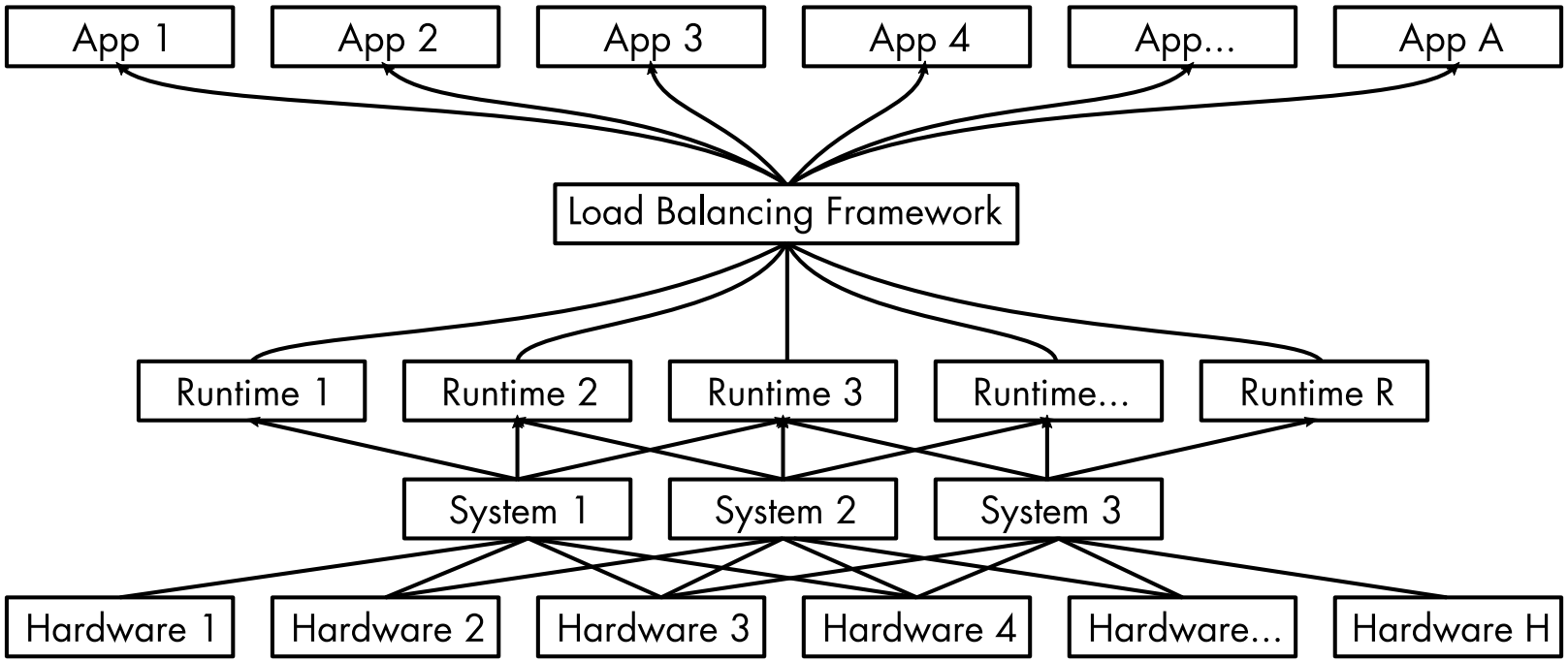




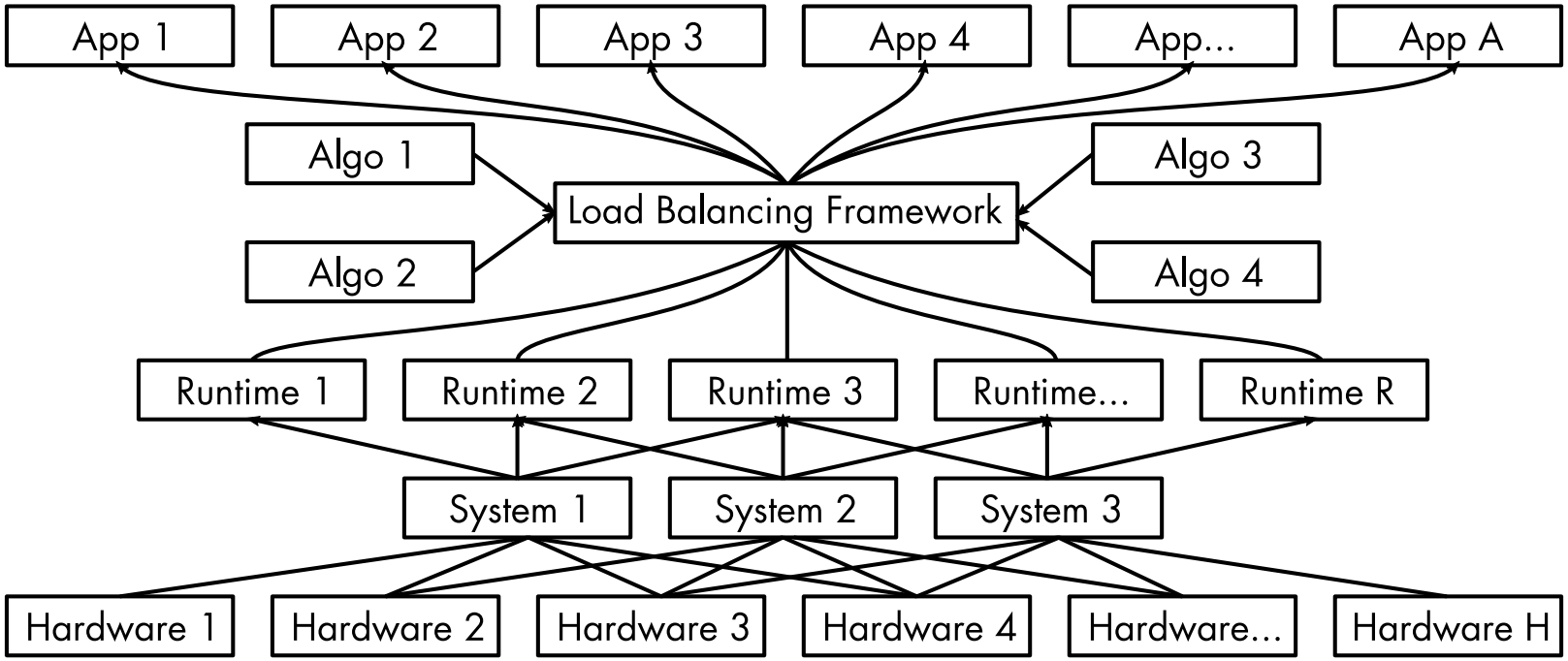
Interfacing



Interfacing



Interfacing



Abstraction

Abstraction

Abstraction

Design of a common abstraction layer

- Language agnostic
- Available resources
- Topology
- Task characteristics
- Task dependency graph

Abstraction

Design of a common abstraction layer

- Language agnostic
- Available resources
- Topology
- Task characteristics
- Task dependency graph

Resources

- CPU/GPU/Accelerator model
- Frequency
- Cache size
- Instruction set
- Thermal Design Power
- Total memory
- Memory usage
- Node-to-node effective bandwidth
- ...

Abstraction

Design of a common abstraction layer

- Language agnostic
- Available resources
- Topology
- Task characteristics
- Task dependency graph

Resources

- CPU/GPU/Accelerator model
- Frequency
- Cache size
- Instruction set
- Thermal Design Power
- Total memory
- Memory usage
- Node-to-node effective bandwidth
- ...

Tasks

- Task type
- Associated data
- Dependency graph
- Task execution time statistics
- Memory usage statistics
- ...

Abstraction

Abstraction

Global abstract cost function

$$f_{cost} = \sum_p w_p \times \lambda_p = w_{time} \times \lambda_{time} + w_{memory} \times \lambda_{memory} + w_{power} \times \lambda_{power} + \dots$$

Abstraction

Global abstract cost function

$$f_{cost} = \sum_p w_p \times \lambda_p = w_{time} \times \lambda_{time} + w_{memory} \times \lambda_{memory} + w_{power} \times \lambda_{power} + \dots$$

Local abstract constraints

$$g_{i \in N_{node}, p} < C_p$$

Abstraction

Global abstract cost function

$$f_{cost} = \sum_p w_p \times \lambda_p = w_{time} \times \lambda_{time} + w_{memory} \times \lambda_{memory} + w_{power} \times \lambda_{power} + \dots$$

Local abstract constraints

$$g_{i \in N_{node}, p} < C_p$$

Summary

Optimization problem: resources + task graph + cost function/constraints

Abstraction

Global abstract cost function

$$f_{cost} = \sum_p w_p \times \lambda_p = w_{time} \times \lambda_{time} + w_{memory} \times \lambda_{memory} + w_{power} \times \lambda_{power} + \dots$$

Local abstract constraints

$$g_{i \in N_{node}, p} < C_p$$

Summary

Optimization problem: resources + task graph + cost function/constraints

Specification of a generic load balancing framework

- As generic as possible
- As simple as possible
- As portable as possible
- As high performance as possible

Abstraction

Global abstract cost function

$$f_{cost} = \sum_p w_p \times \lambda_p = w_{time} \times \lambda_{time} + w_{memory} \times \lambda_{memory} + w_{power} \times \lambda_{power} + \dots$$

Local abstract constraints

$$g_{i \in N_{node}, p} < C_p$$

Summary

Optimization problem: resources + task graph + cost function/constraints

Specification of a generic load balancing framework

- As generic as possible
- As simple as possible
- As portable as possible
- As high performance as possible

Challenge

Genericity vs Usability vs Portability vs Performance

Standardization

Standardization

De jure or de facto standards: a working approach

- MPI
- OpenMP
- BLAS
- ...

Standardization

De jure or de facto standards: a working approach

- MPI
- OpenMP
- BLAS
- ...

Strategy

Produce a specification, not an implementation

Standardization

De jure or de facto standards: a working approach

- MPI
- OpenMP
- BLAS
- ...

Strategy

Produce a specification, not an implementation

Implementation

A reference implementation should just serve as a nice addition and as a demonstrator

Standardization

De jure or de facto standards: a working approach

- MPI
- OpenMP
- BLAS
- ...

Strategy

Produce a specification, not an implementation

Implementation

A reference implementation should just serve as a nice addition and as a demonstrator

Advantages

- Sustainability, interoperability, portability
- Language agnosticism
- Long term maintainability
- Specification act as documentation for implementers
- Allow many competing implementations with specific optimizations

Strategy

General idea

- Start from existing frameworks and libraries
- Look for common patterns
- Design abstractions based on these common patterns
- Find convoluted use cases that do not fit in these abstractions
- Make abstractions more generic
- Repeat until abstractions become more convoluted than the use cases

Strategy

General idea

- Start from existing frameworks and libraries
- Look for common patterns
- Design abstractions based on these common patterns
- Find convoluted use cases that do not fit in these abstractions
- Make abstractions more generic
- Repeat until abstractions become more convoluted than the use cases

A possible approach

- Write a white paper describing requirements (`www.load-balancing.fr`)
- Design abstractions
- Write a specification
- Write an implementation based on this specification
- Make the specification an ISO standard (eg: C++ Standards Committee)

The conceptification of load balancing

Concept-based programming

- Coming in C++20 (`-fconcepts` in `GCC`)
- A way to describe interfaces through requirements on template parameters
- A variable always has a single type, but a type can belong to several concepts

Example

```
1  // Concept definitionb
2  template <class T>
3  concept Addable = requires (T a, T b) {
4      a + b;
5  };
6
7  // Concept use
8  template <Addable T>
9  constexpr T add(T a, T b) noexcept {
10     return a + b;
11 }
```

Strategy

Start with the design of a library of concepts instead of a library of types and algorithms.

Parallel programming in C++

Parallel programming in C++

C++11

- Atomics
- Threads
- Mutexes
- Futures

Parallel programming in C++

C++11

- Atomics
- Threads
- Mutexes
- Futures

C++17

- Execution policies
- Standard parallel algorithms

Parallel programming in C++

C++11

- Atomics
- Threads
- Mutexes
- Futures

C++17

- Execution policies
- Standard parallel algorithms

C++20

- Coroutines
- *(Concepts)*

Parallel programming in C++

C++11

- Atomics
- Threads
- Mutexes
- Futures

C++17

- Execution policies
- Standard parallel algorithms

C++20

- Coroutines
- *(Concepts)*

C++23

- Executors
- Networking
- Transactional memory
- *(Reflection)*

Conclusion

Conclusions

Conclusions

Main goal: abstracting load balancing

Makes it far easier to prototype, design, implement, and deploy load balancing algorithms through common abstractions.

Conclusions

Main goal: abstracting load balancing

Makes it far easier to prototype, design, implement, and deploy load balancing algorithms through common abstractions.

Main research directions

- Focus on abstraction
- Produce a specification
- Aim for standardization
- Write a reference implementation (first in terms of concepts)

Conclusions

Main goal: abstracting load balancing

Makes it far easier to prototype, design, implement, and deploy load balancing algorithms through common abstractions.

Main research directions

- Focus on abstraction
- Produce a specification
- Aim for standardization
- Write a reference implementation (first in terms of concepts)

Collaboration

- Come and talk to me!
- Contact: vreverdy@illinois.edu
- Participate: www.load-balancing.fr

Thank you for your attention

`www.load-balancing.fr`