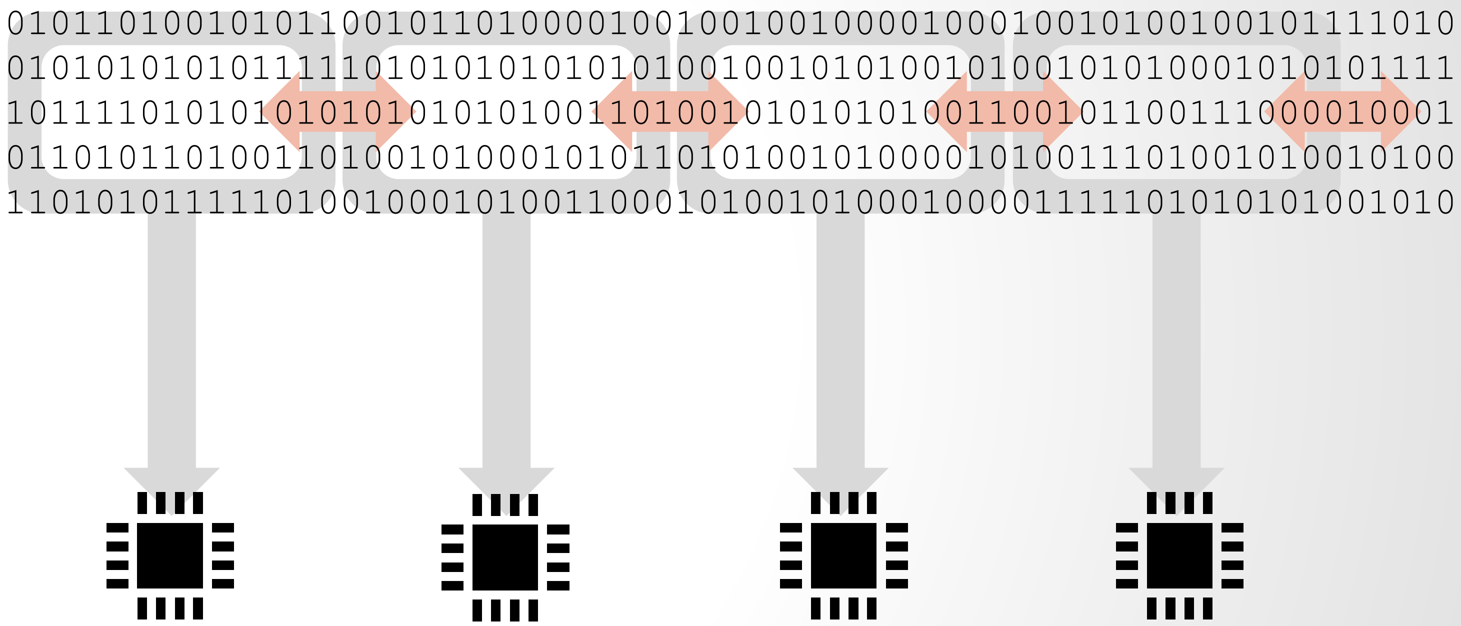
proThOS

Programmierung und Ausführung von taskbasierten HPC Anwendungen

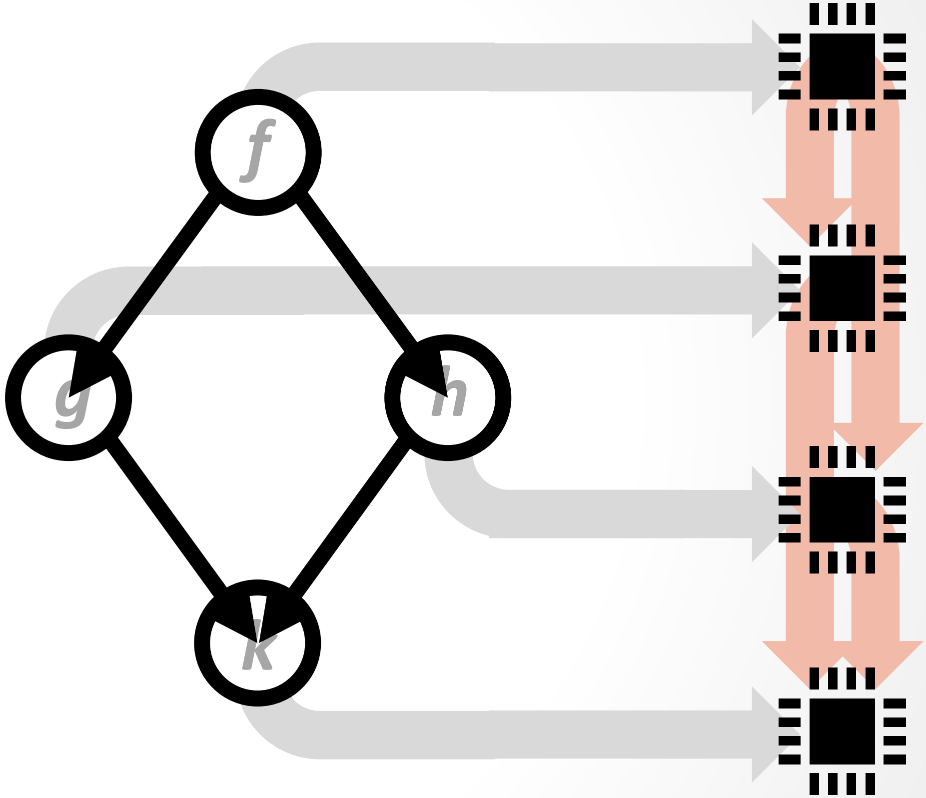
Lutz Schubert
lutz.Schubert@uni-ulm.de

ulm university universität uulm

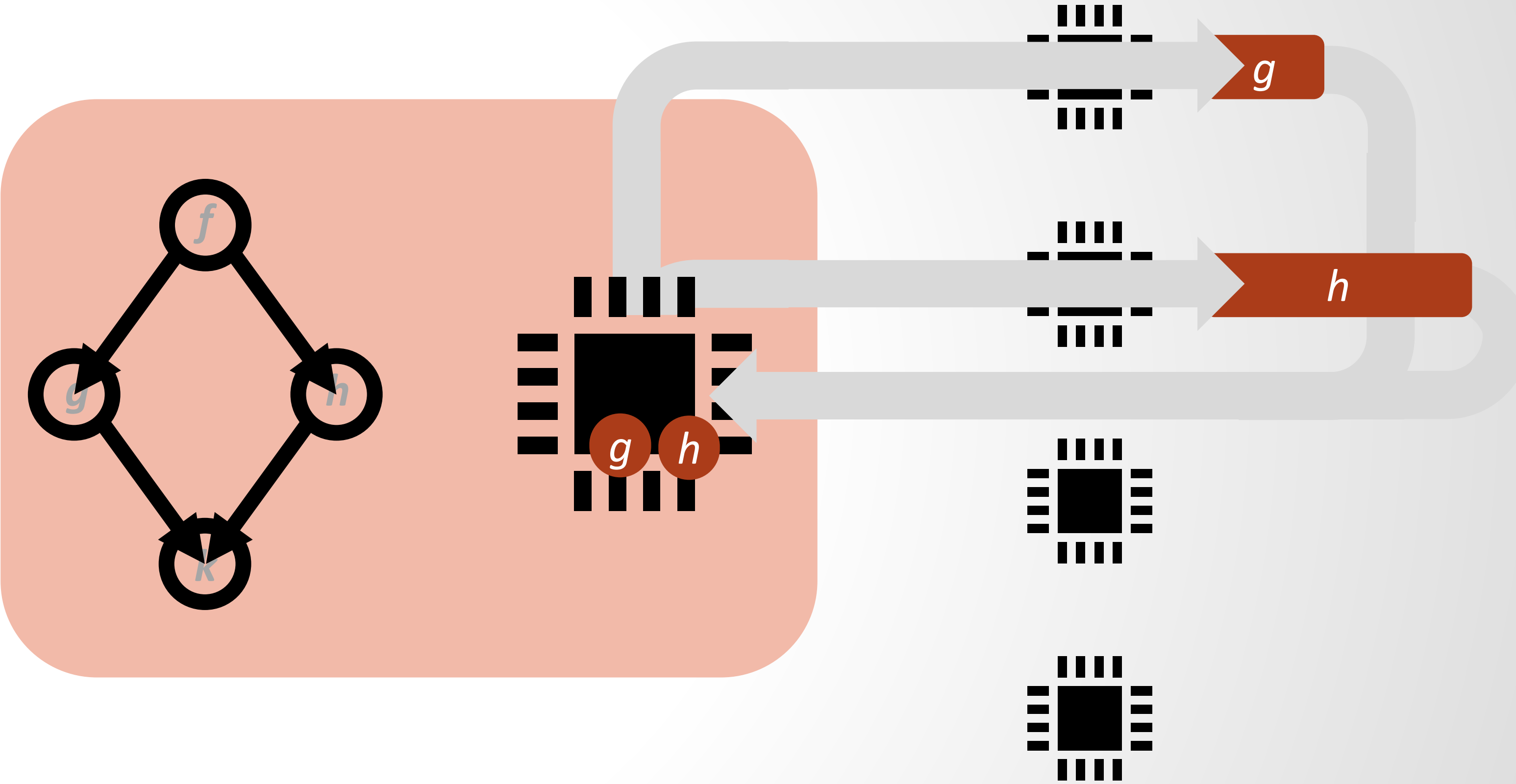
Data Parallelism



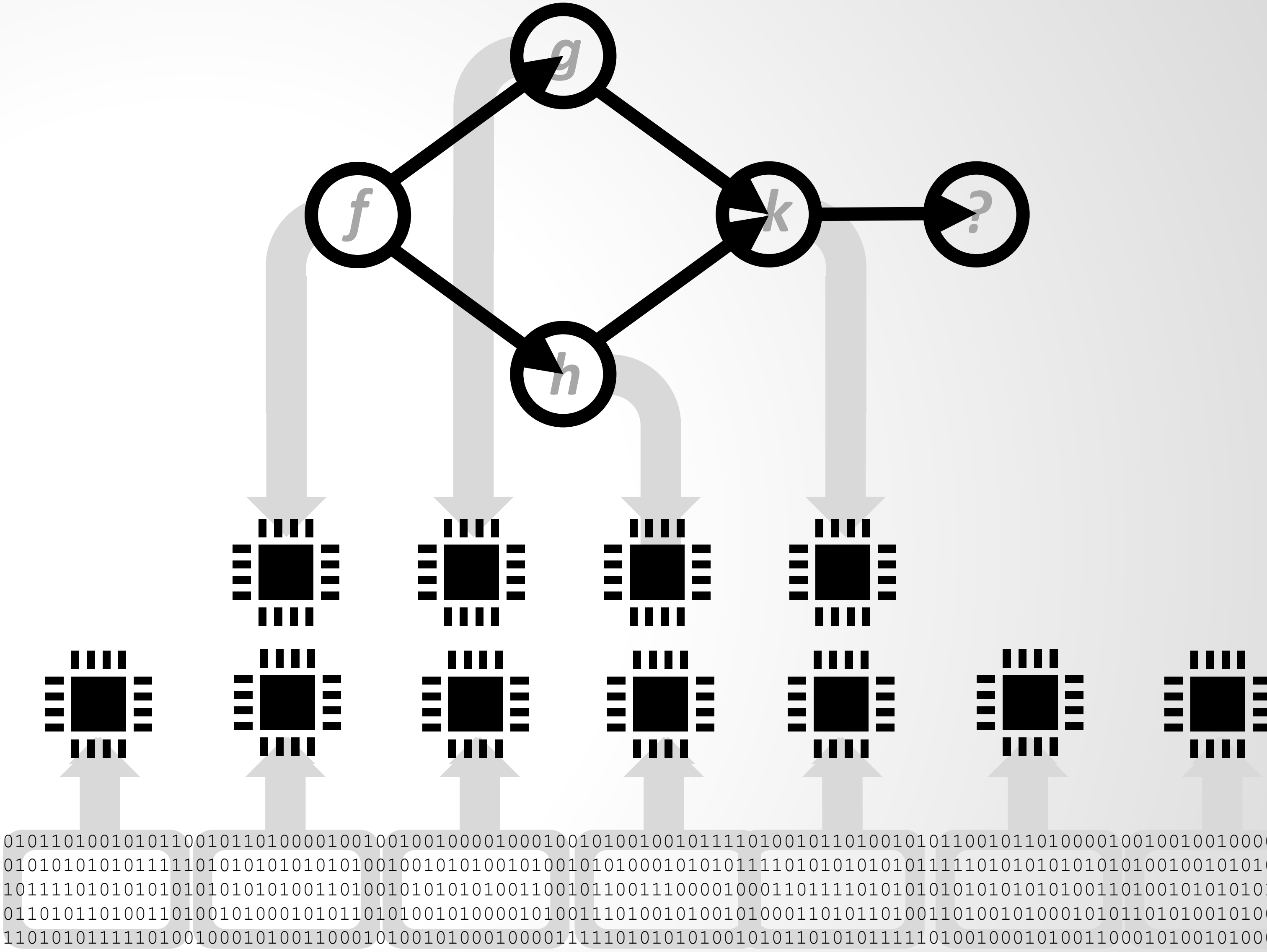
Task-Based Parallelism



Task- vs. Data-Based Parallelism
Overhead



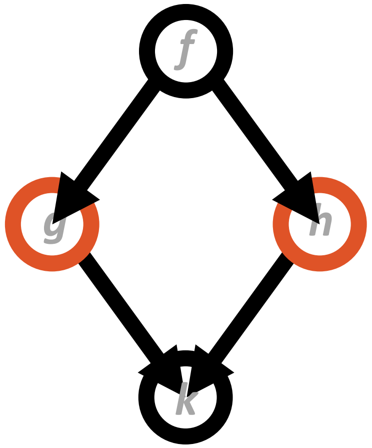
Task- vs. Data-Based Parallelism Scale

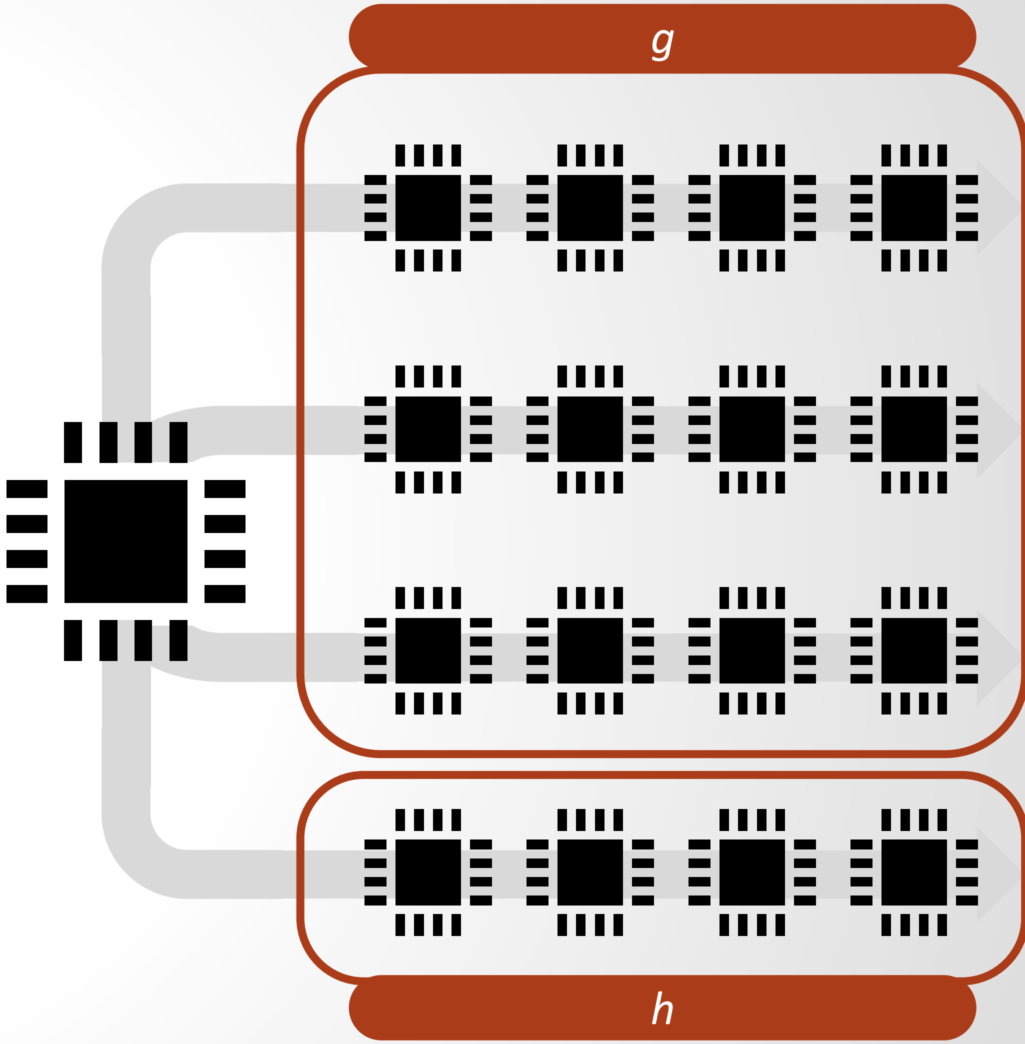


Task- vs. Data-Based Parallelism

- All parallelisation generates overhead
- In general, overhead for task-based parallelism is higher than for data-based
  - Scheduling
  - Data gathering
  - Identification of next tasks
  - …
- Most algorithms show a higher degree of data parallelism than task independency

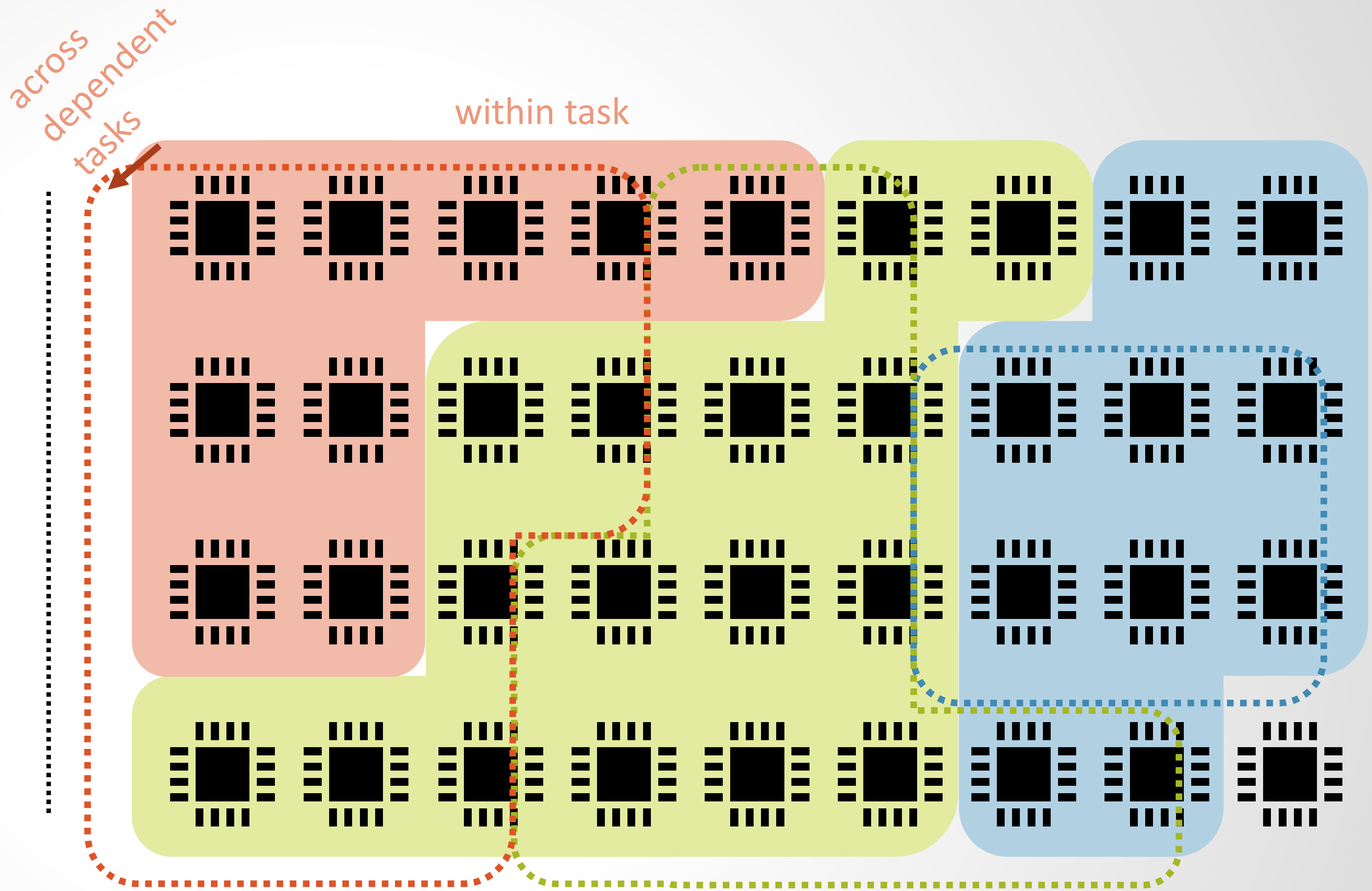
Mixed Parallelism





Mixed Parallelism

Maintaining Data Locality



Mixed Parallelism

Problems

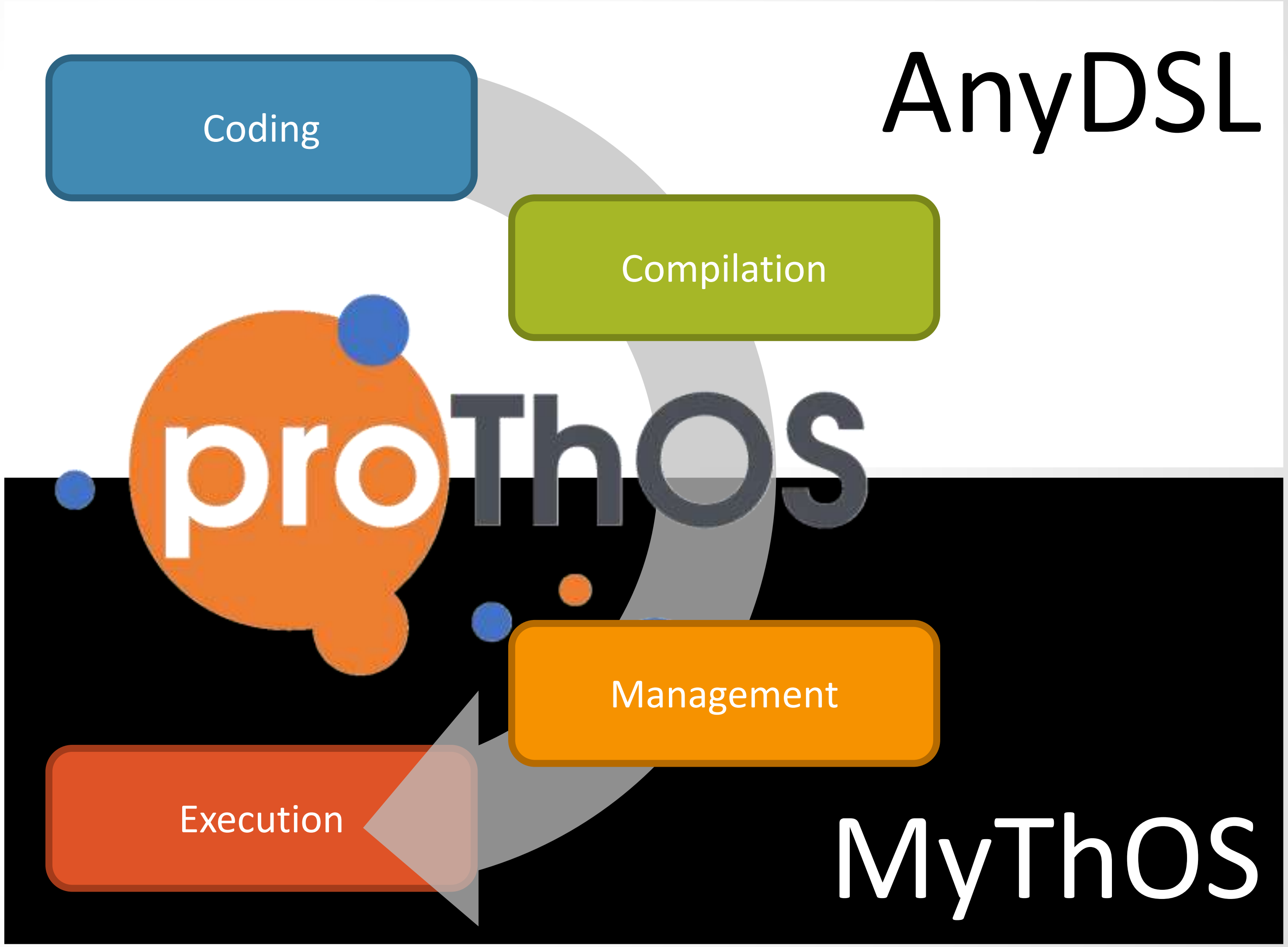
- Complete restructuring of code is necessary
- Data dependencies need to be known
- Overhead must be minimal to allow for management of small tasks and threads

Challenges for ProThOS

- Coding
  - explicit tasks
  - data dependencies
  - parallelism
- Compilation
  - task descriptions
  - communication
  - heterogeneity
- Management
  - scheduling
  - data locality
  - data availability
- Execution
  - task initialization overhead
  - communication overhead
  - memory management

General Scope

Two Major Building Blocks



Two Major Building Blocks

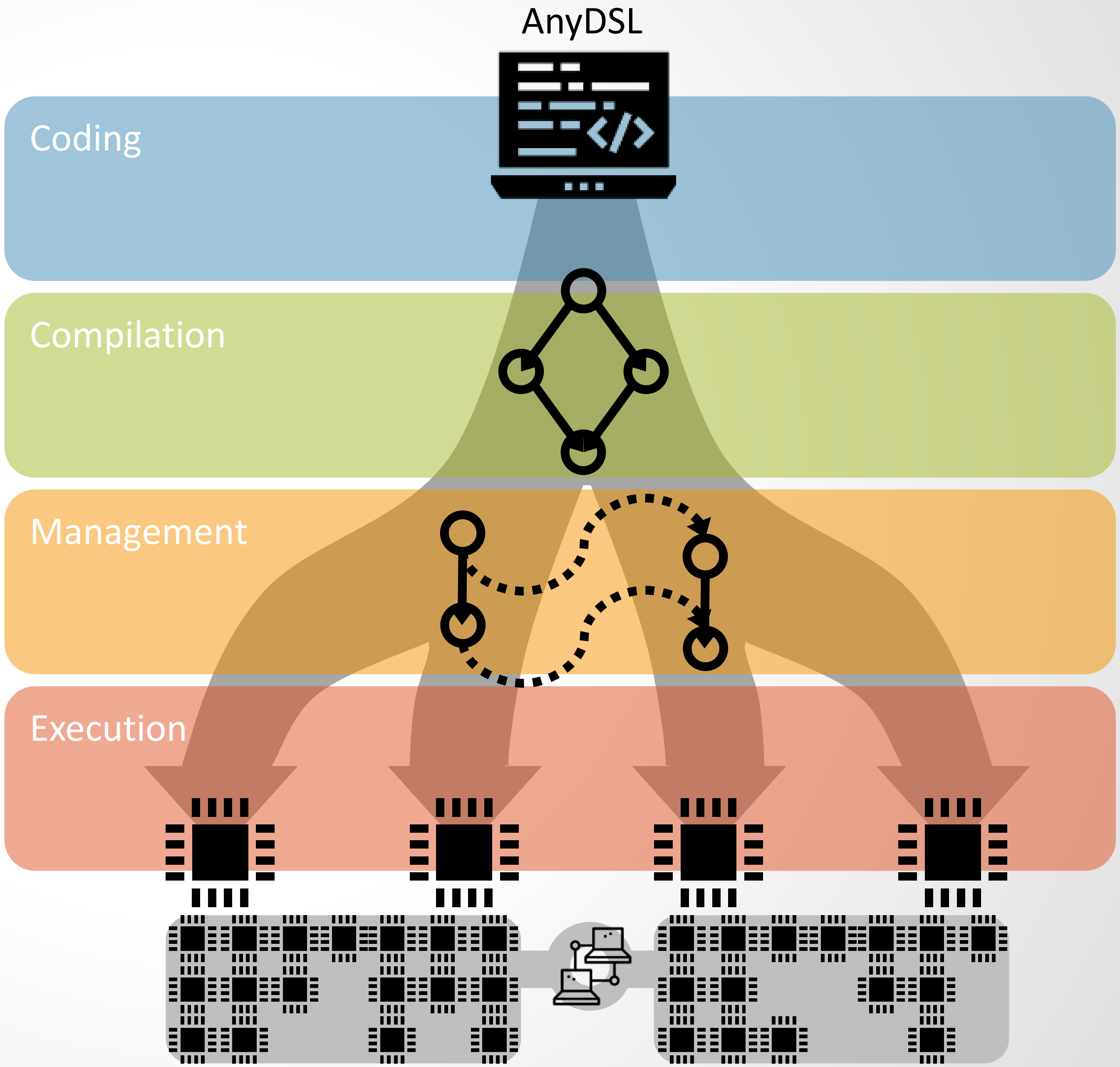
AnyDSL

- Developed by DFKI & Uni Saarbrücken
- Allows to generate DSLs
- Highly efficient compilation
- Specifically developed for Vectorization

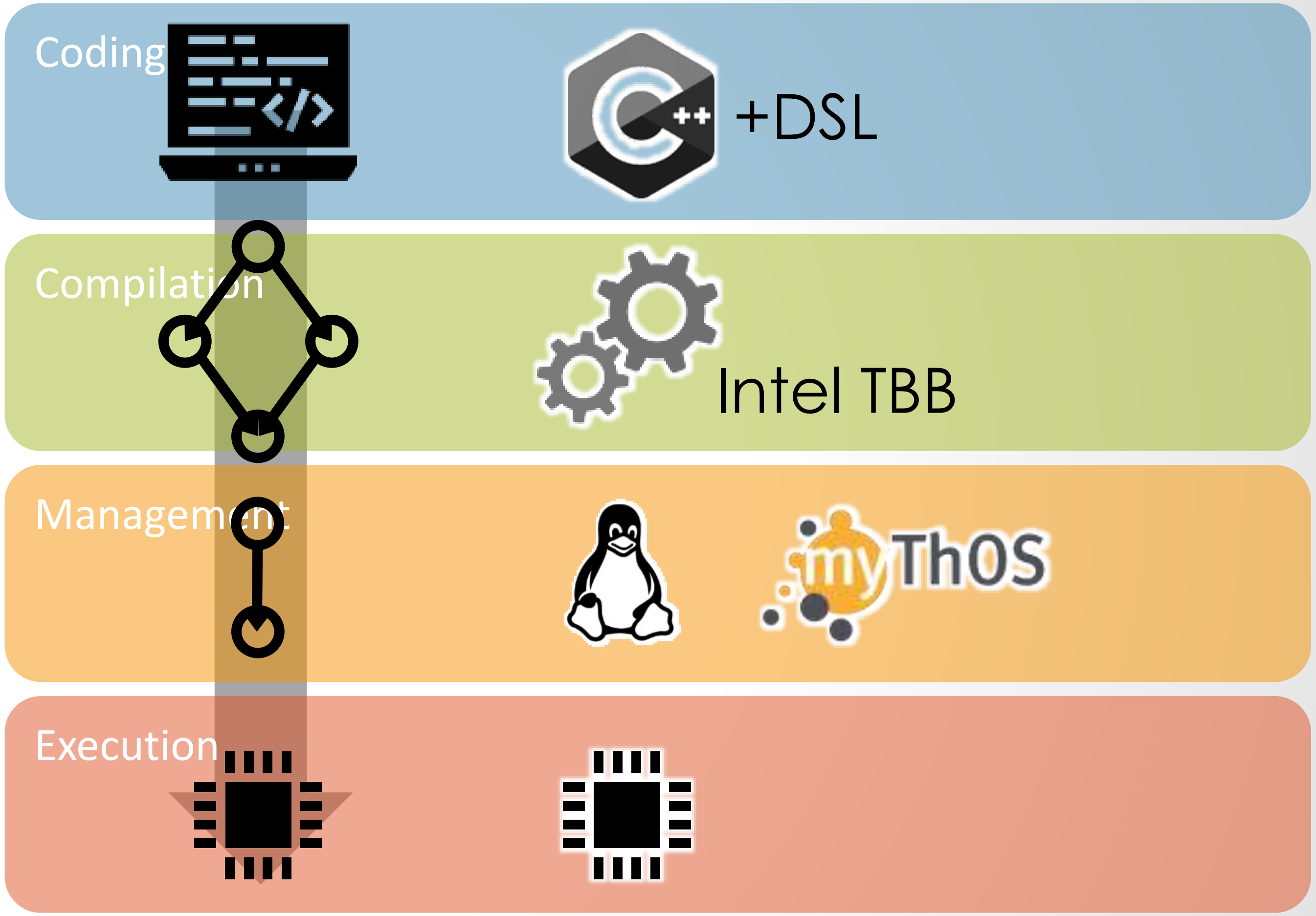
MyThOS

- Developed by BTU Cottbus and Uni Ulm
- Minimal, modular kernel
- Allows for easy extension with new functionalities
- Much better pThread management than Linux

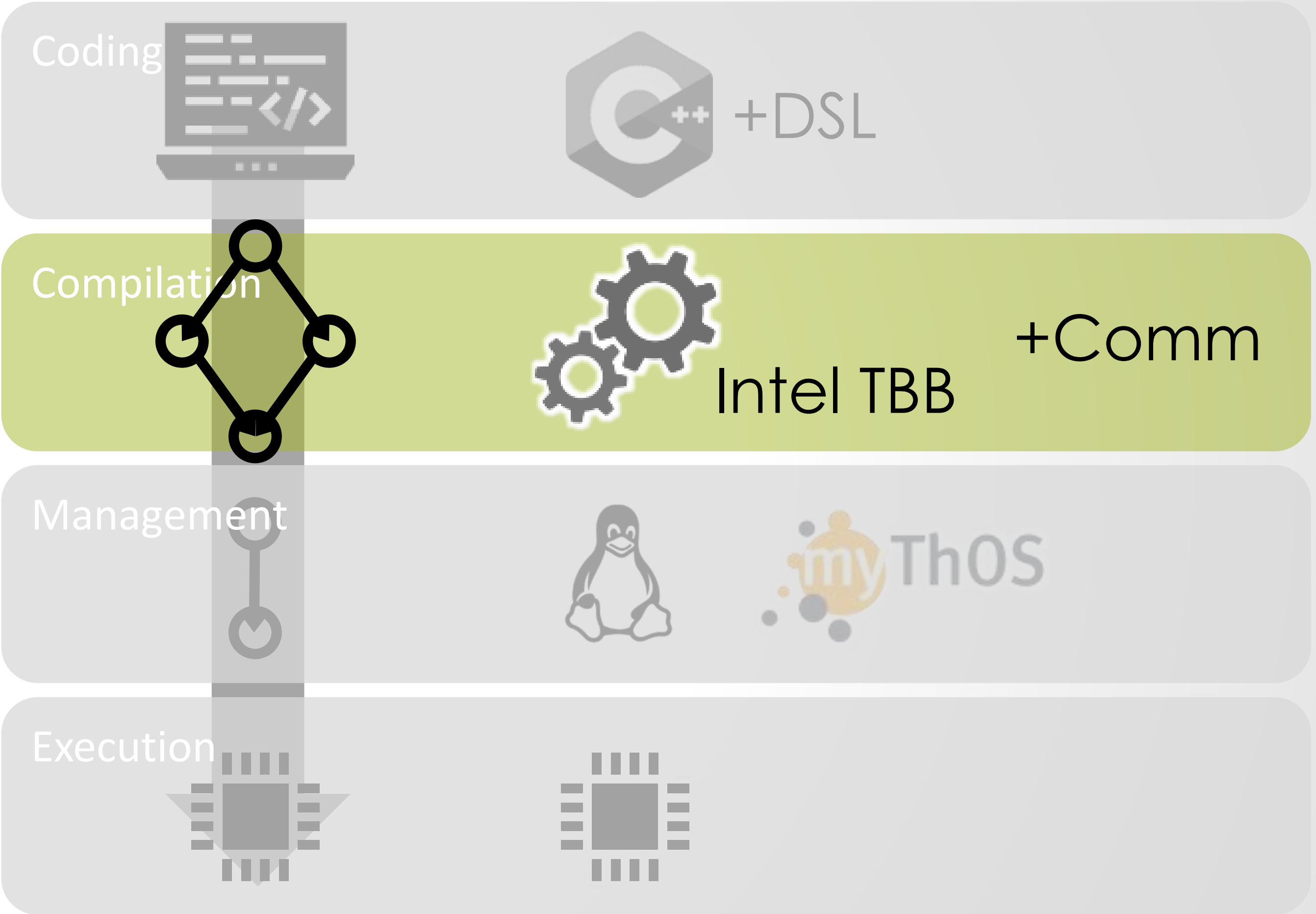
ProThOS Lifecycle



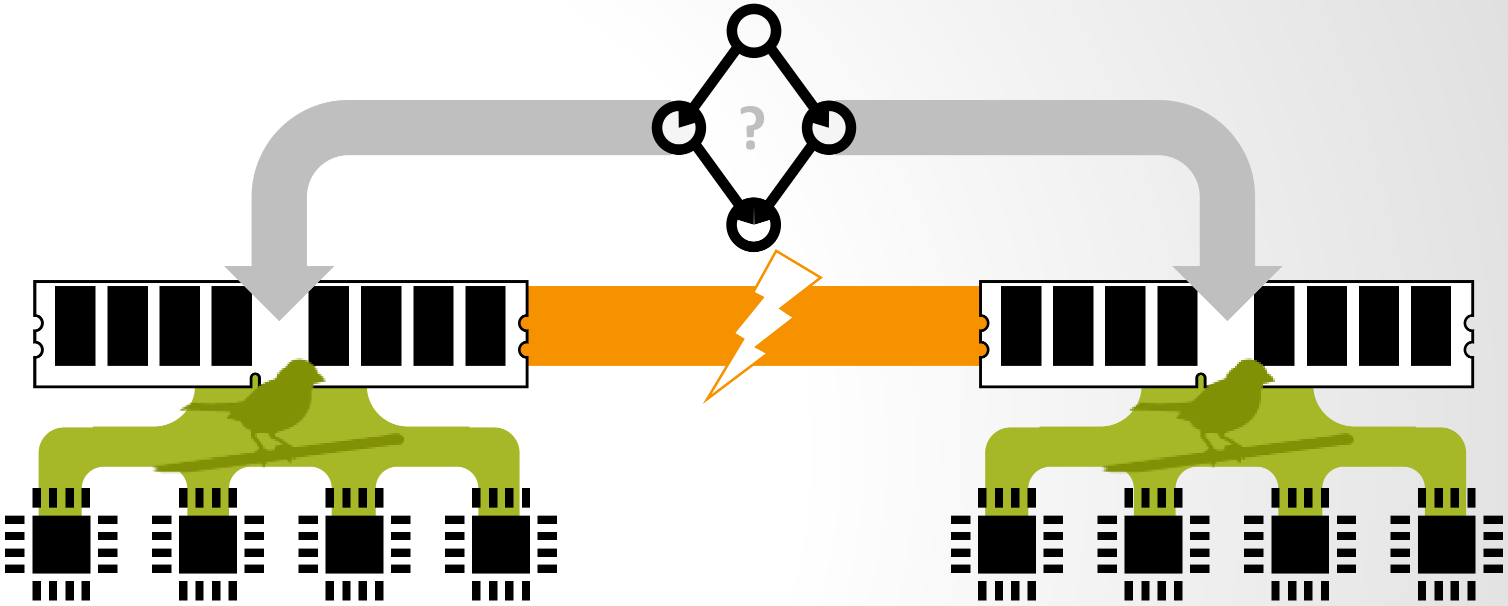
Principle Technology Support



Actual Technology & Status
Intel TBB

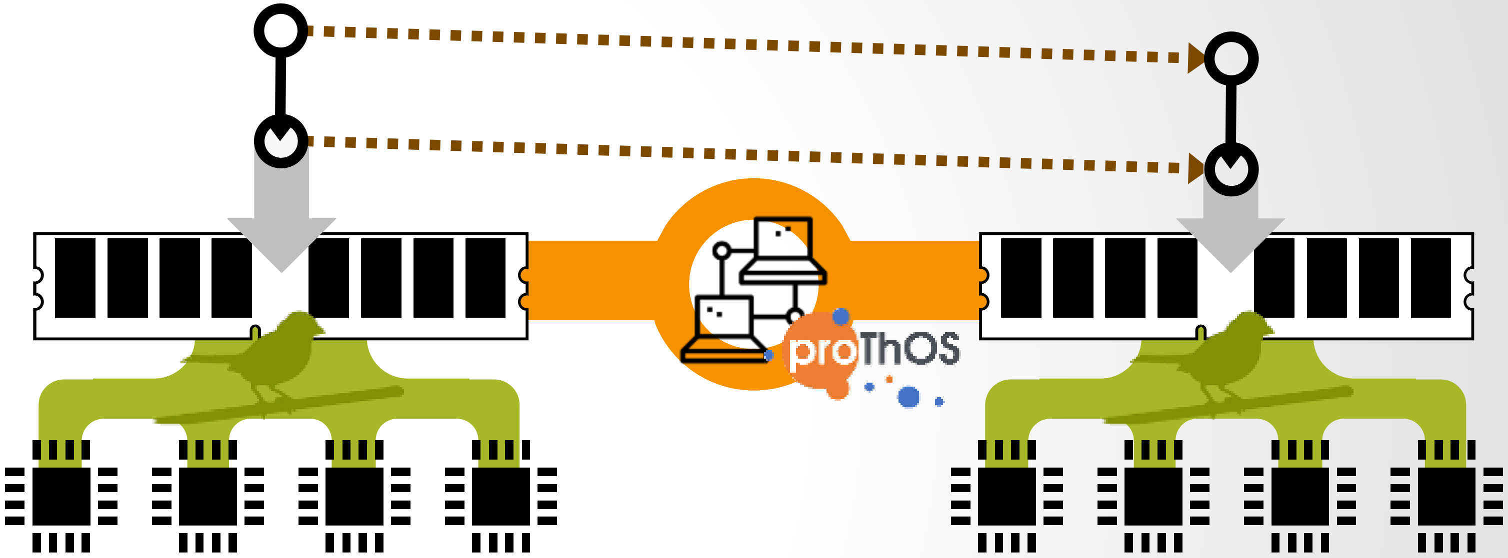


Actual
Technology &
Status
Task
Description



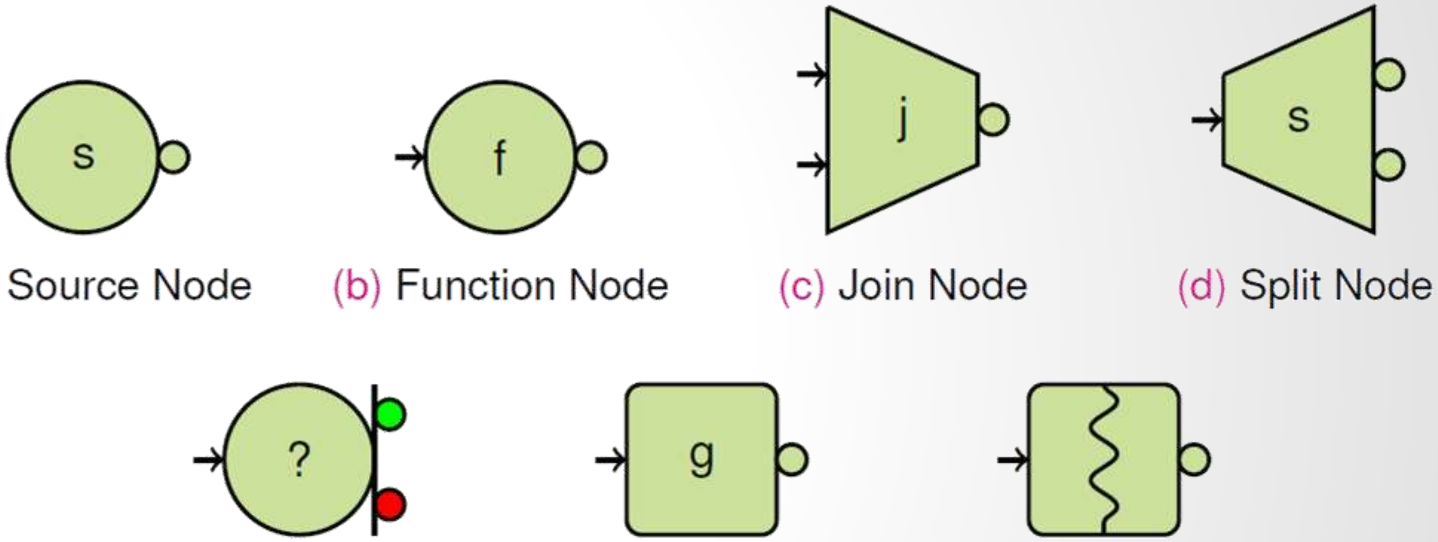
Actual
Technology &
Status

Task
Description



Actual Technology & Status

Node Types

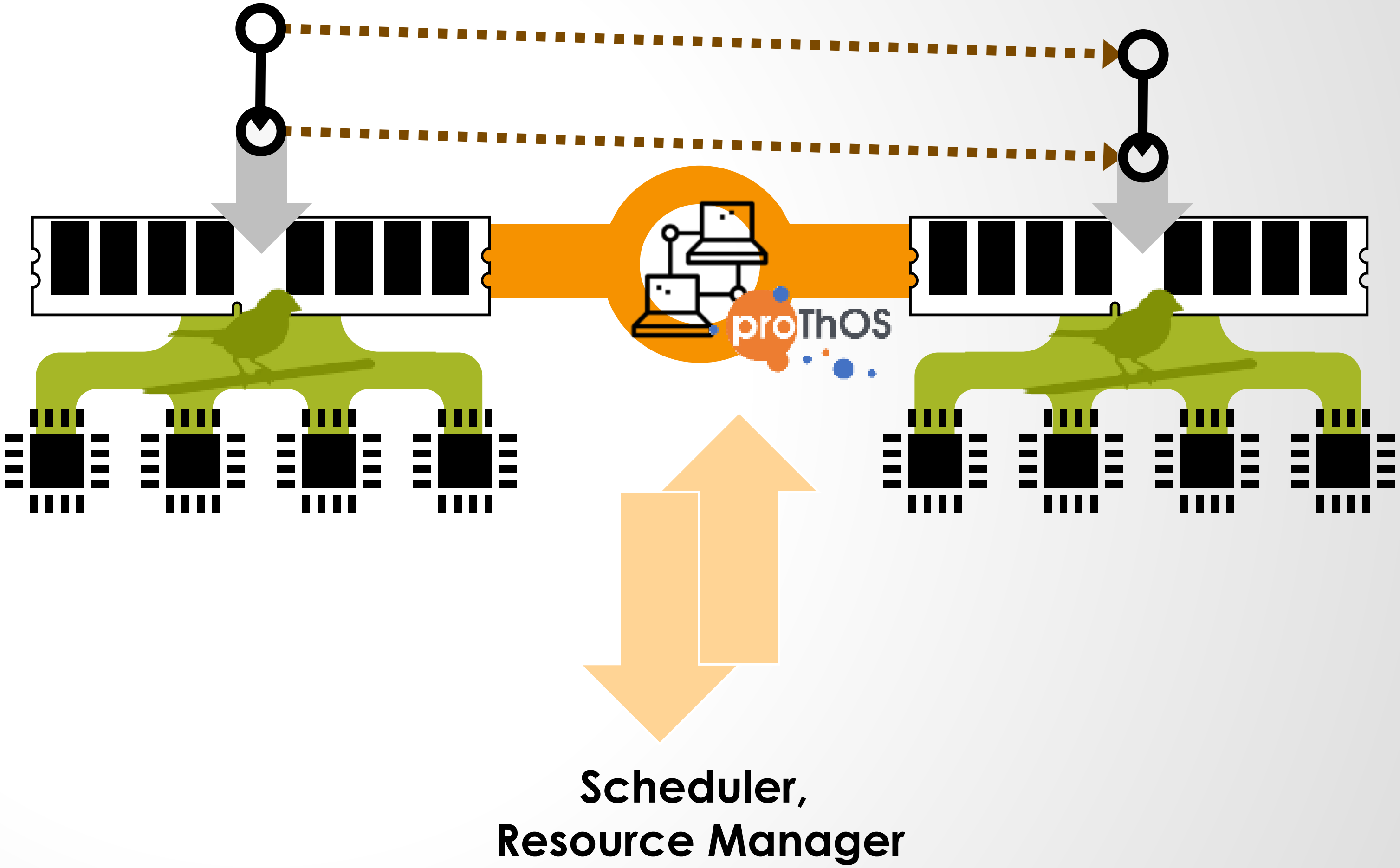


(a) Source Node (b) Function Node (c) Join Node (d) Split Node

(e) Conditional Node (f) Subgraph Node (g) Async Node

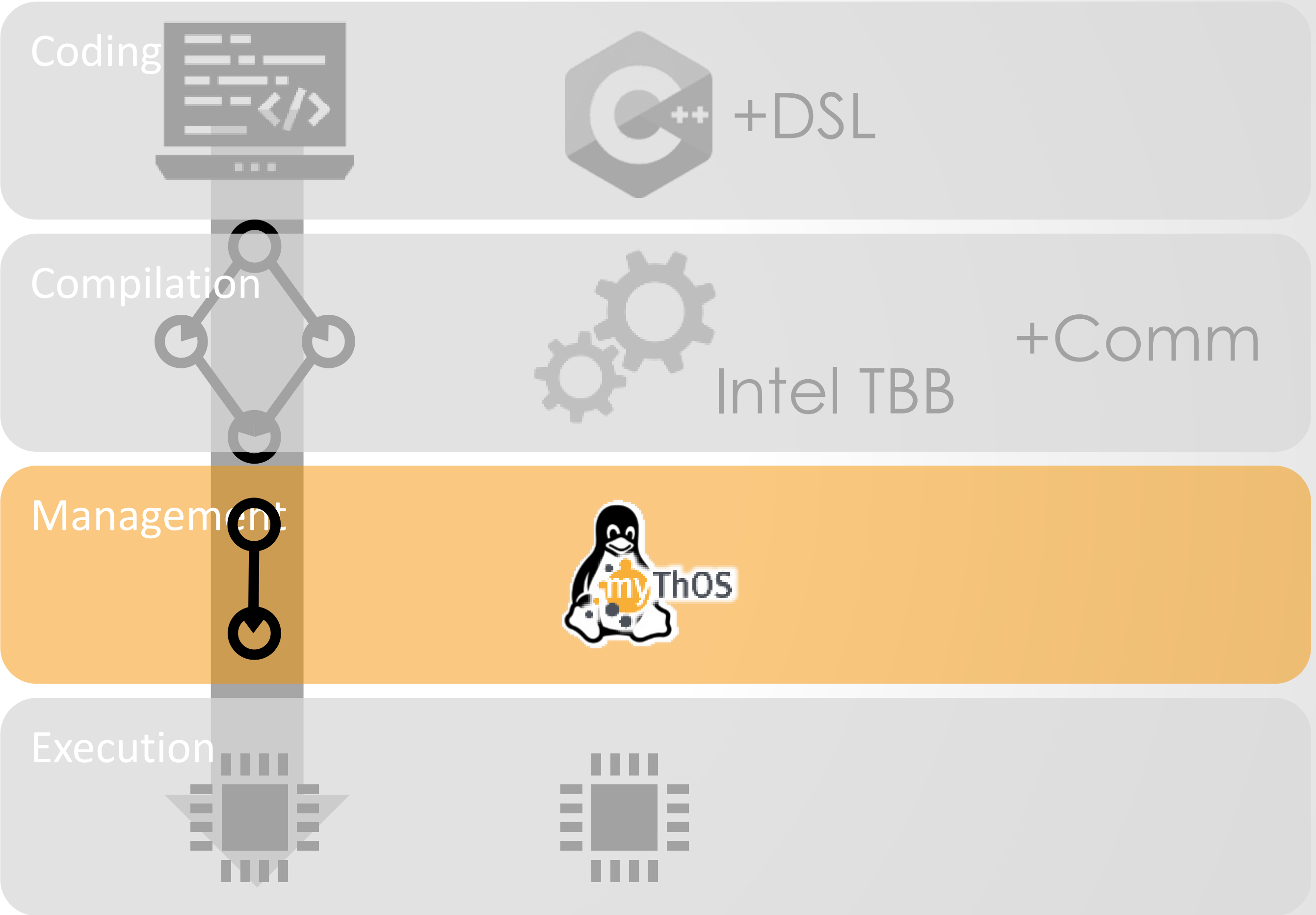
Actual Technology & Status

Task Description



Actual Technology & Status

MyThOS + Linux



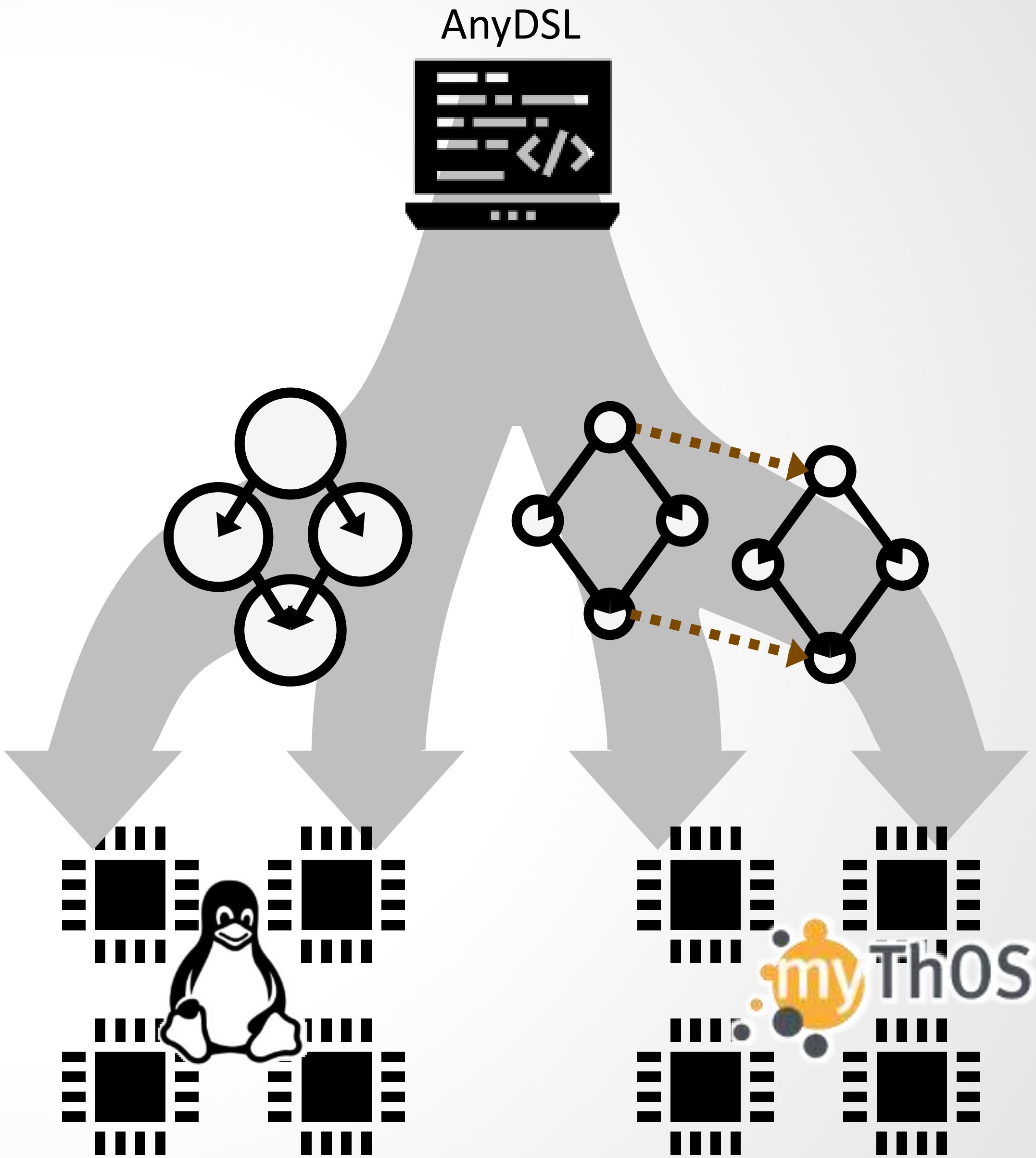
Actual Technology & Status MyThOS + Linux

- monolithic
- creates considerable overhead
- not designed for scalability and HPC
- more difficult to adjust to new environments
- but has a network driver ;)

- minimal, modular OS
- provides up to 20x faster thread management than Linux => smaller tasks possible
- offers distributed memory management
- and distributed scheduling
- still missing network(!)

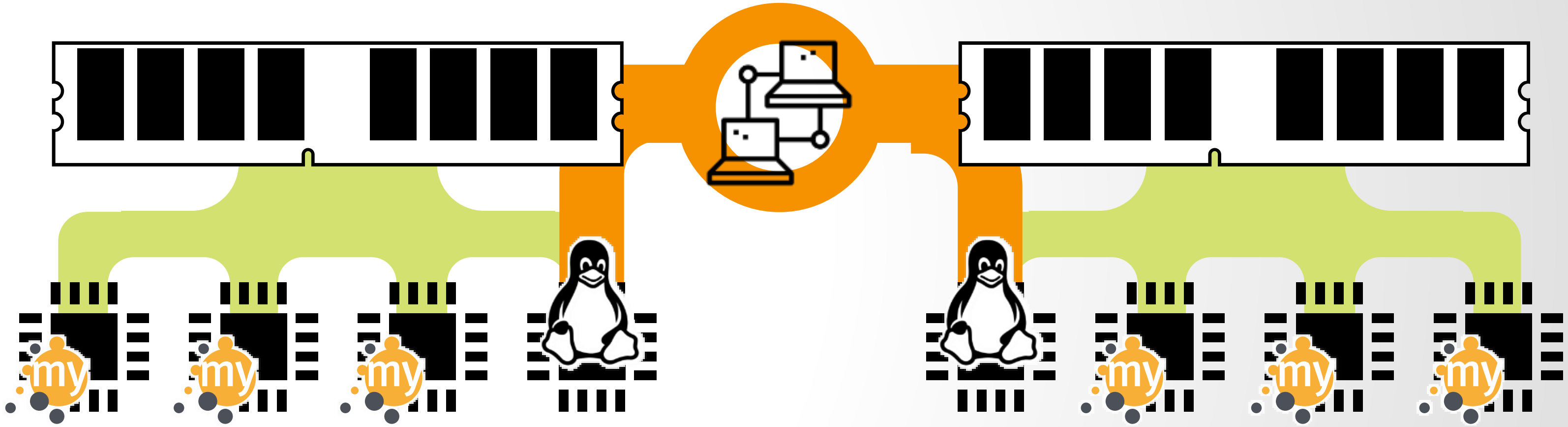
Actual Technology & Status

MyThOS + Linux



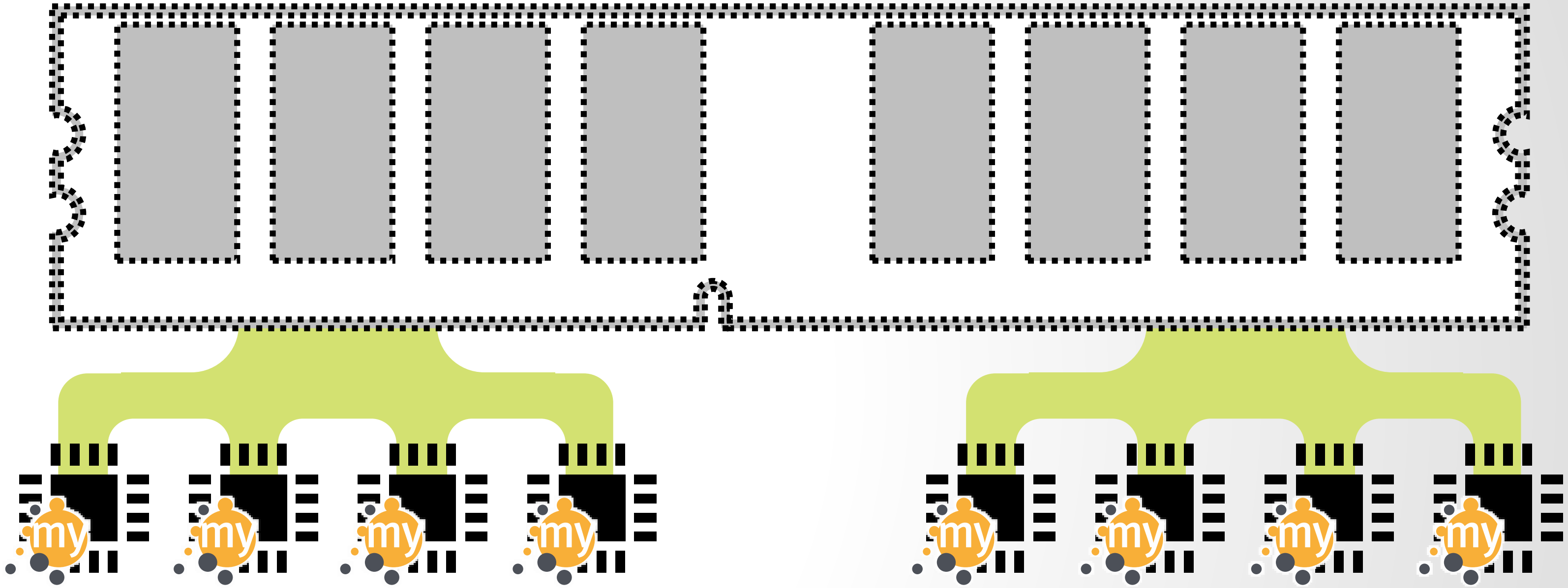
Actual Technology & Status

MyThOS + Linux



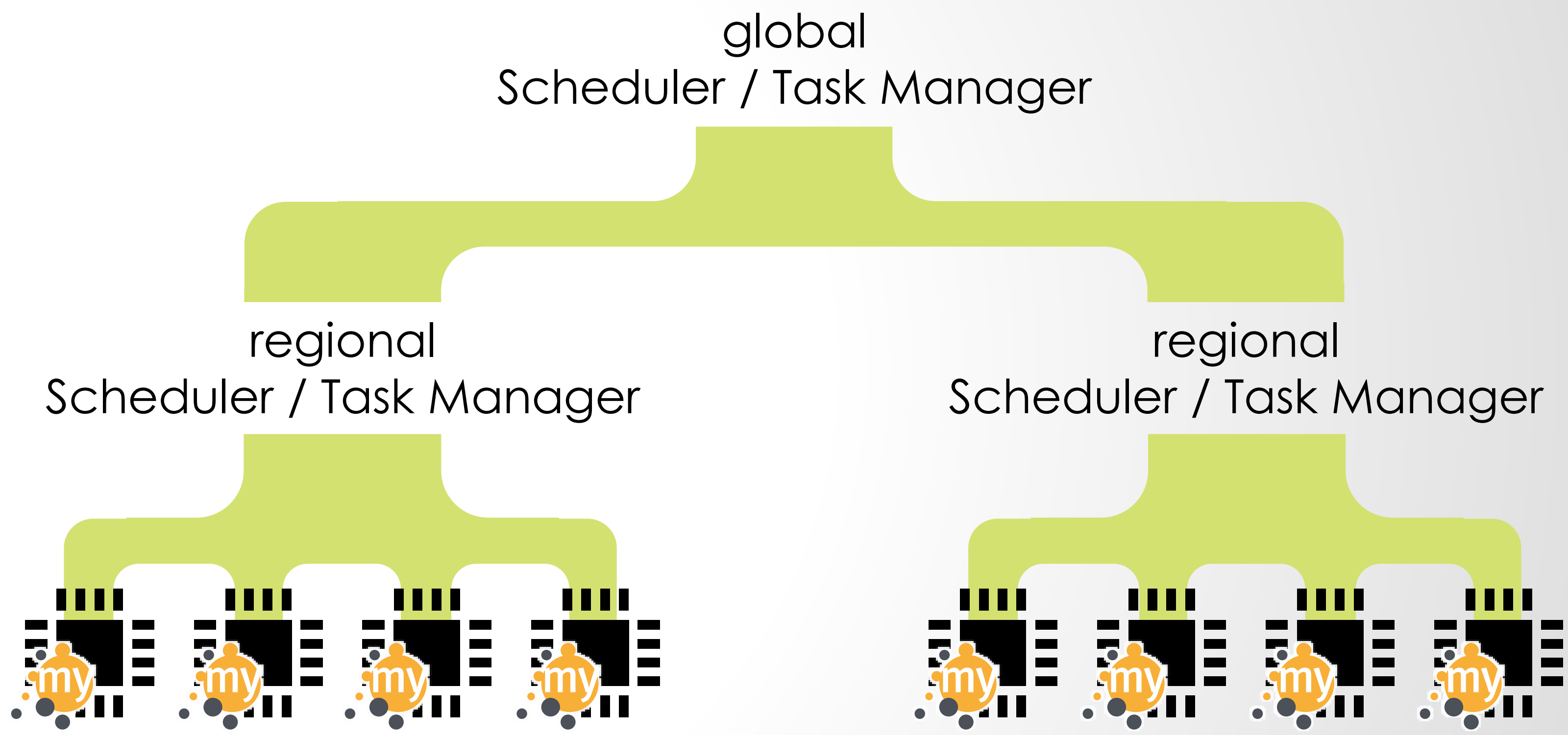
Actual Technology
& Status

Virtually Shared Distributed Memory



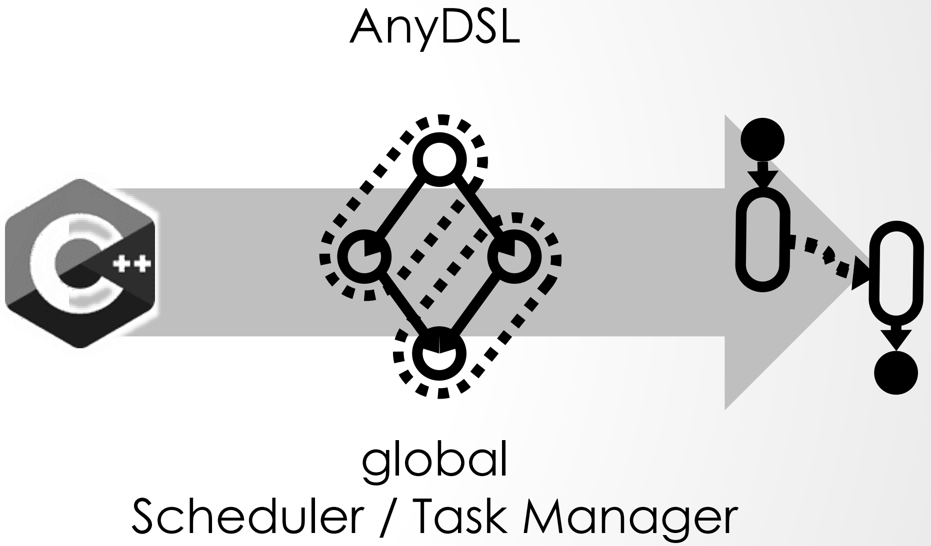
Actual Technology
& Status

Hierarchical Scheduling



Actual
Technology &
Status

Flowgraph



Actual Technology & Status

Flowgraph Unrolling

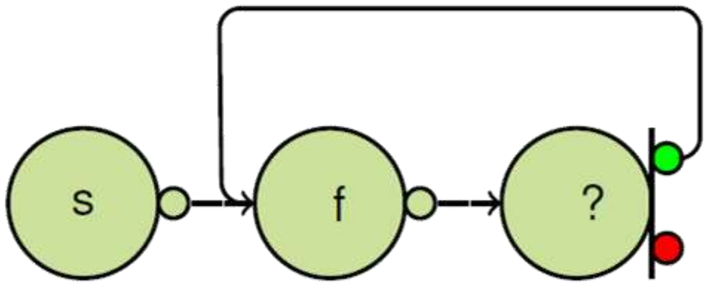


Figure: Flow Graph example

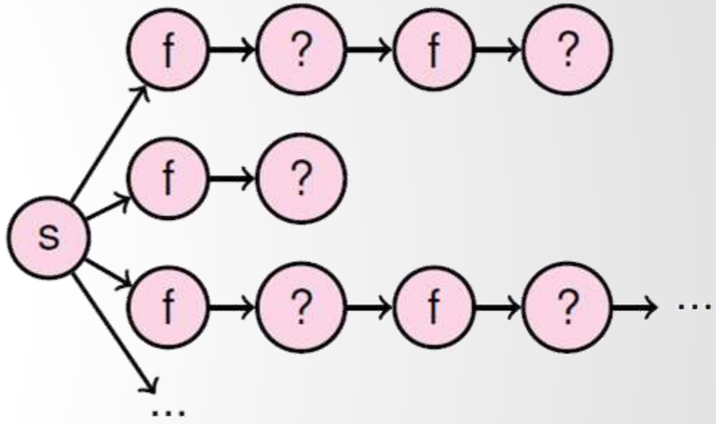
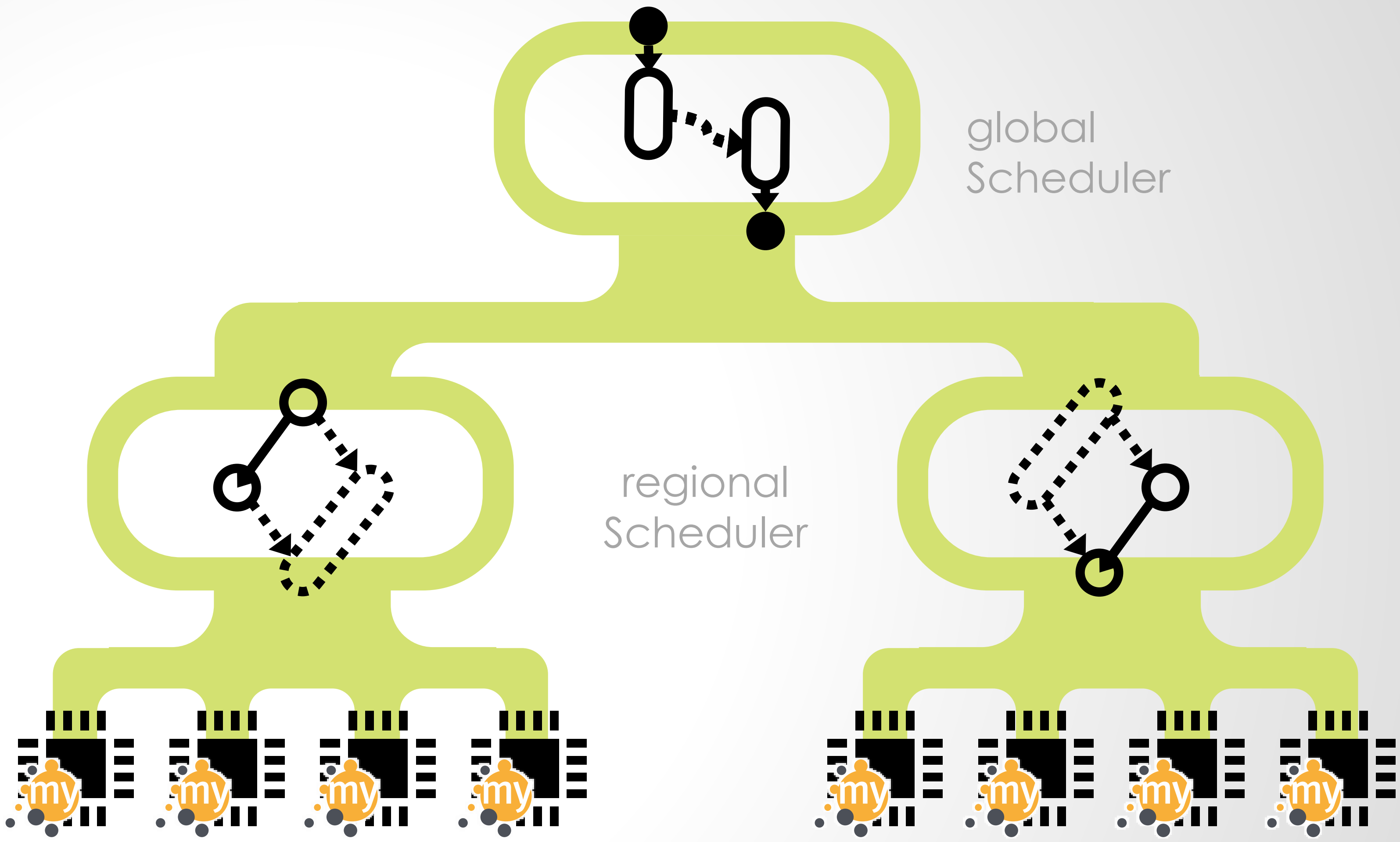


Figure: Unrolled Task Graph (DAG)

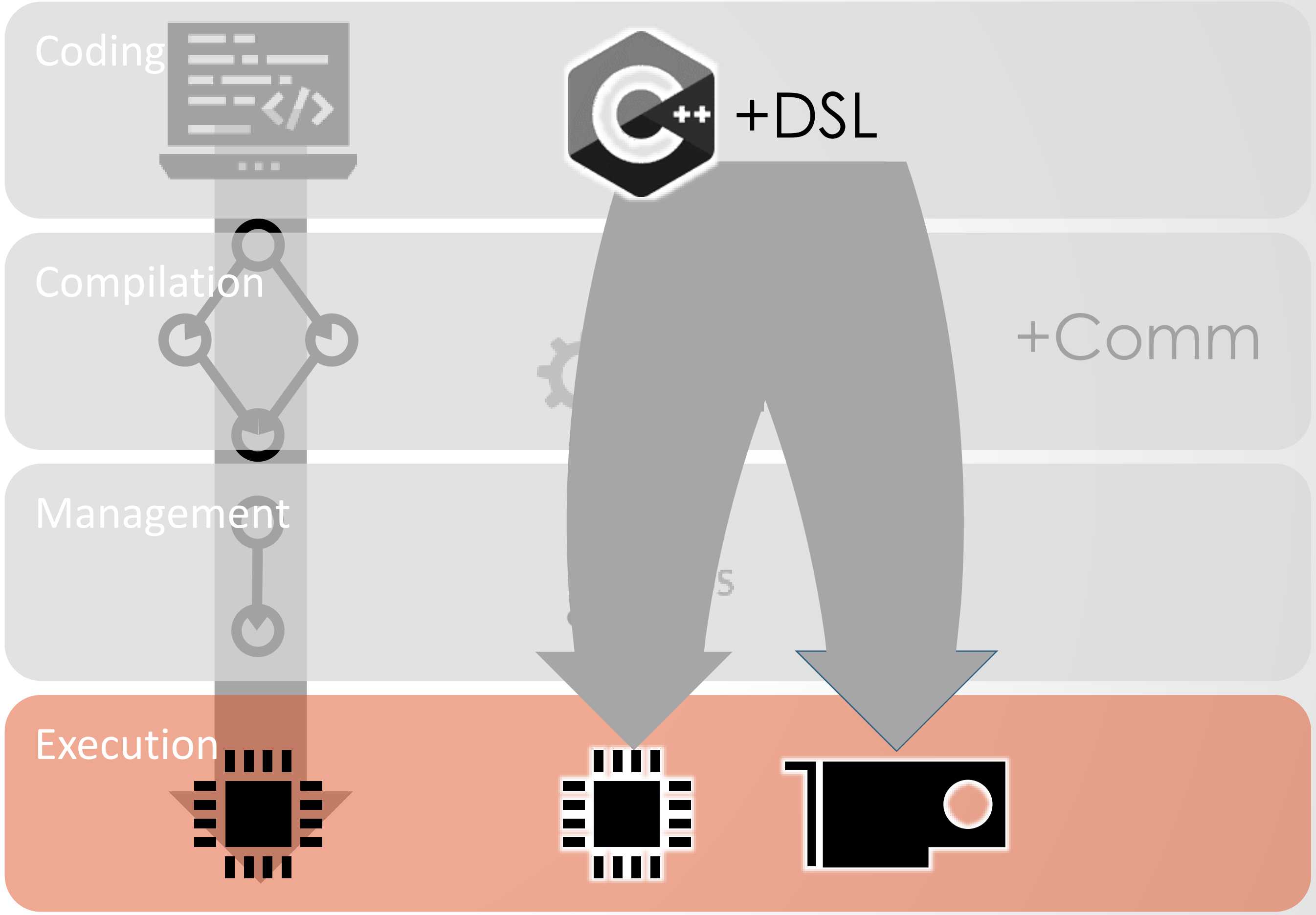
Actual
Technology &
Status

Flowgraph Scheduling

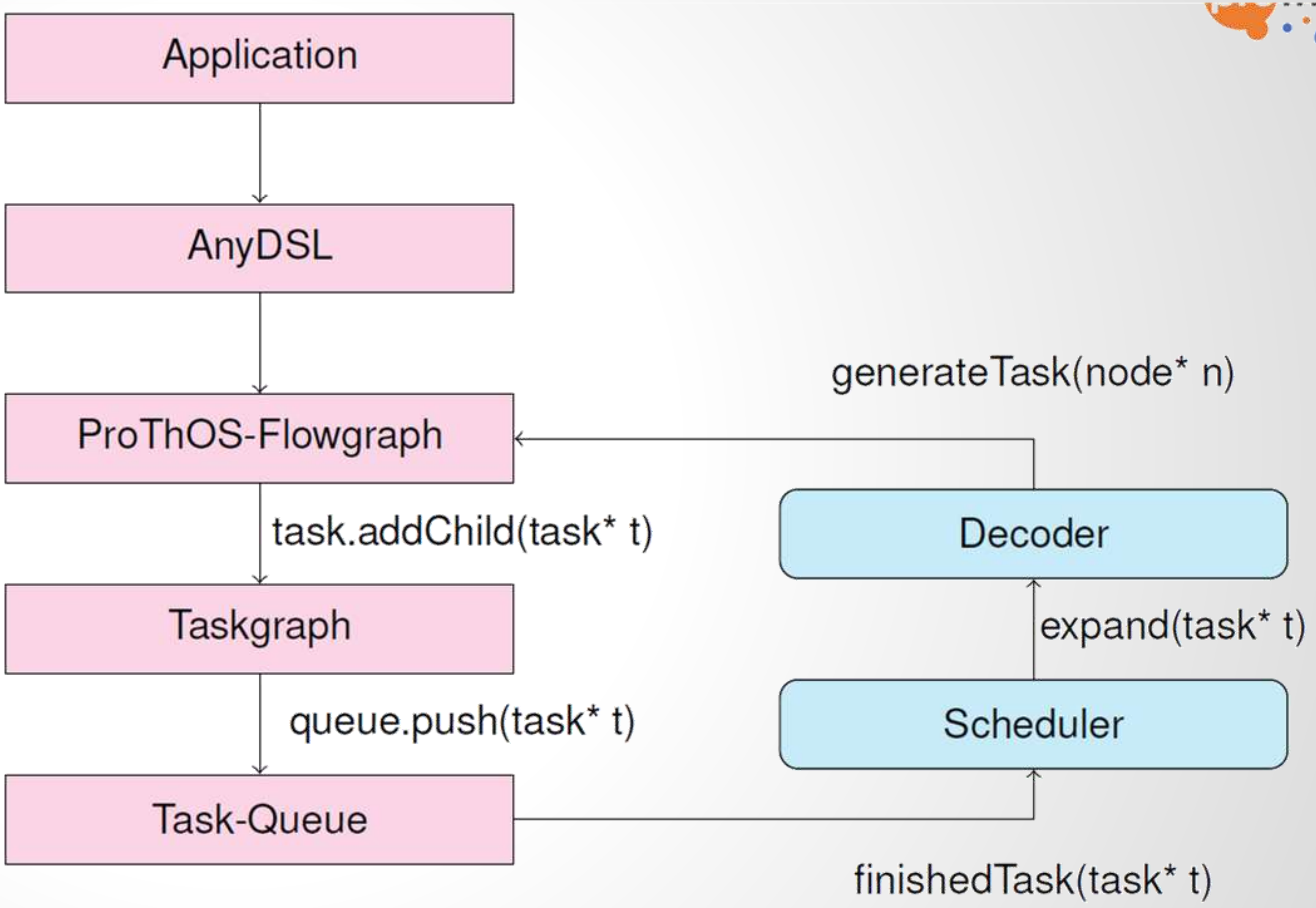


Actual Technology & Status

Destination Resources



ProThOS Lifecycle Overview

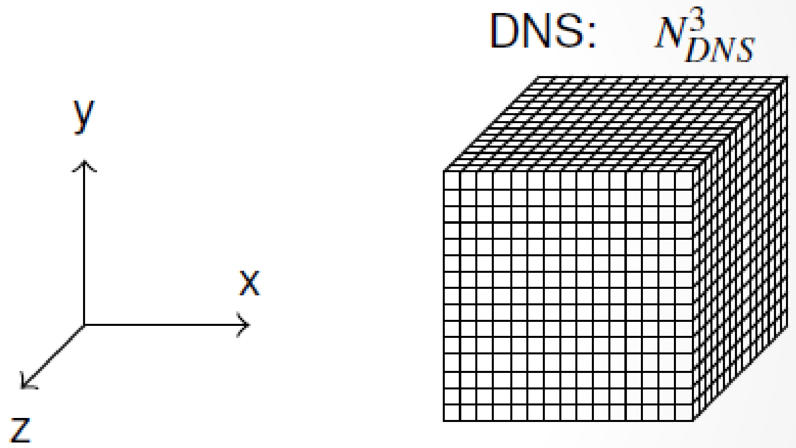


Use Case Scenario

Example

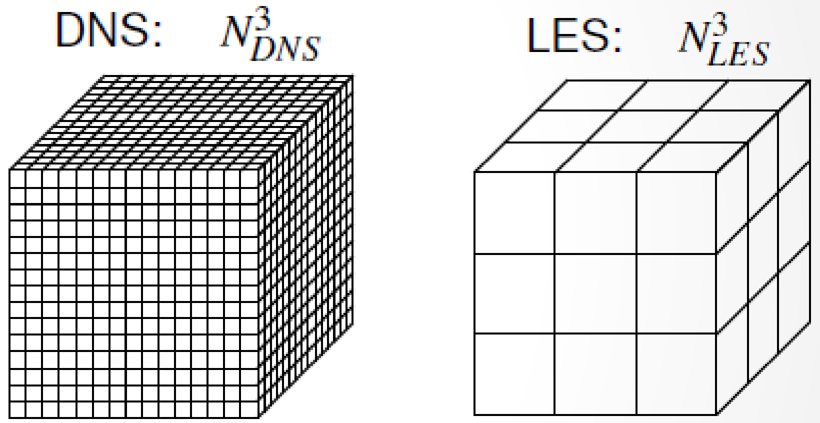
ODTLES
Überlagerte Gitter

- Navier-Stokes-Gleichung $\frac{\partial \mathbf{V}}{\partial t} + \nabla(\mathbf{V} \circ \mathbf{V}) = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{V}$



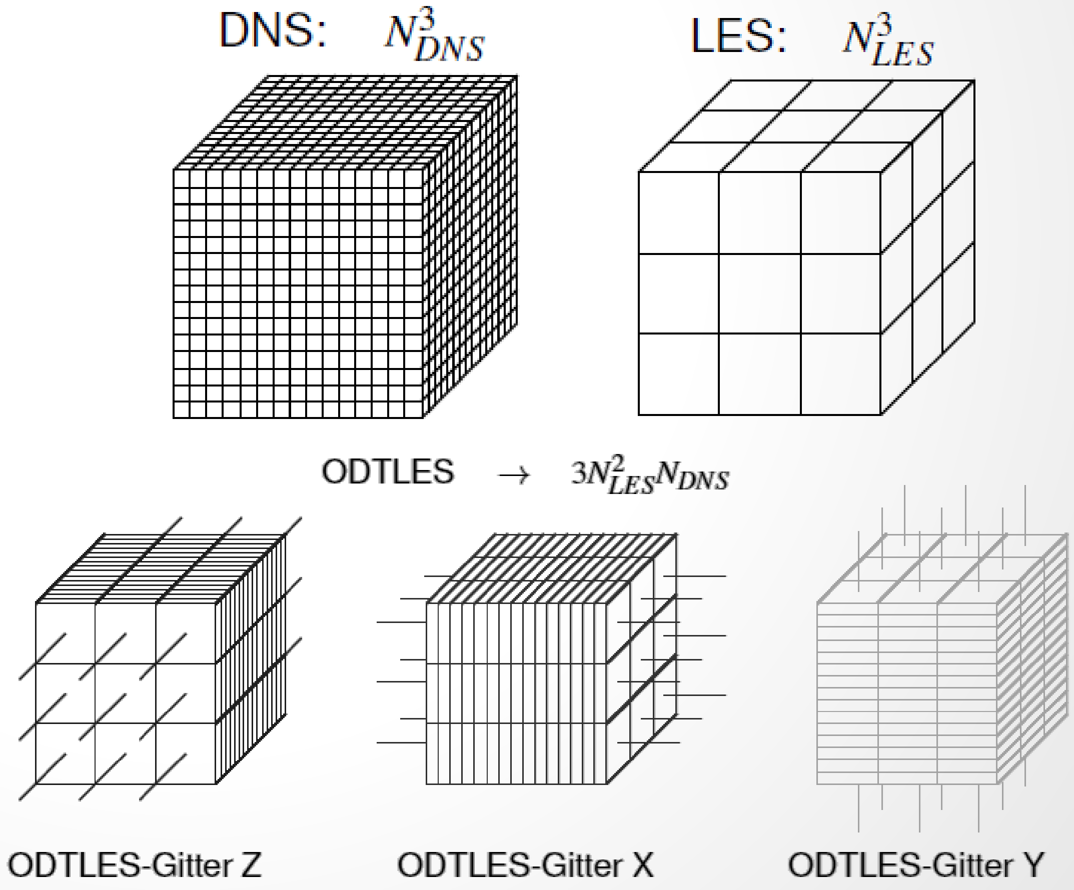
ODTLES
Überlagerte Gitter

- Navier-Stokes-Gleichung $\frac{\partial \mathbf{V}}{\partial t} + \nabla(\mathbf{V} \circ \mathbf{V}) = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{V}$
- Großskalige Advektions- und Diffusionseffekte: LES-Gitter



ODTLES
Überlagerte Gitter

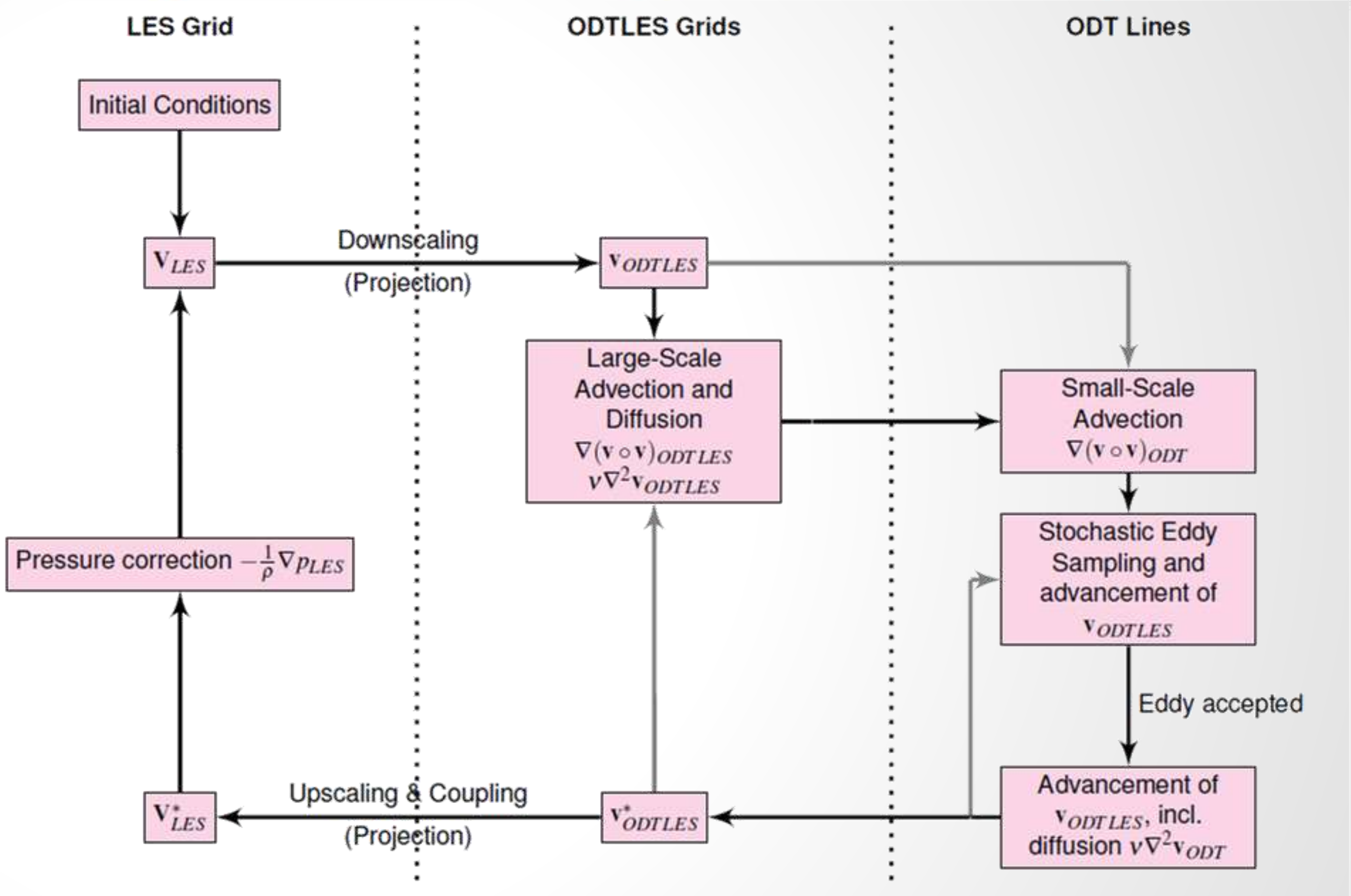
- Navier-Stokes-Gleichung $\frac{\partial \mathbf{V}}{\partial t} + \nabla(\mathbf{V} \circ \mathbf{V}) = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{V}$
- Großskalige Advektions- und Diffusionseffekte: LES-Gitter
- Feinskalige Turbulenz: Advektion entlang 1D Linien in x,y,z-Richtung



ODTLES Algorithmus

1. Eingabe der Anfangsbedingungen (LES)
2. Projektion der LES-Geschwindigkeiten auf die drei ODTLES-Gitter (x,y,z)
3. Bestimmung der großskaligen Advektions- und Diffusionseffekte in jedem ODTLES-Gitter
4. Stochastisches Eddy-Sampling in jeder ODT-Linie
5. Akzeptanz von Wirbeln generiert Beitrag zum zeitlichen Verlauf der ODTLES-Geschwindigkeit
6. Projektion der neuen ODTLES-Geschwindigkeiten auf das LES-Gitter
7. Druckkorrektur der LES-Geschwindigkeiten (Poisson-Problem)
8. Wiederholung von Schritt 2.-7. bis zum Ende der Simulationszeit

ODTLES
Algorithmus



Conclusion

Status and Next Steps

Current Status

- ODTLES:
  - Portierung von Fortran nach C++
  - tw. AnyDSL, Verifikation
- AnyDSL:
  - JIT von C++ aus aufrufen
  - Callbacks mit Closures
  - Interface zu C++ Flow-Graphen, Stincilla Stencil-Pipelines
- Runtime:
  - Ausrollen des Flow-Graph zu azyklischem Taskgraph
- Runtime:
  - Shared-memory Prototyp auf Linux und MyThOS
- MyThOS:
  - libc++ und tw. libc Unterstützung für Anwendungen

Next Steps

- ODTLES:
  - Beschreibung als Flow-Graph
- AnyDSL:
  - Closures mit Parametern
  - Datentransfers im Flow-Graph
- Runtime:
  - Integration zusätzlicher Knotentypen in den Flowgraph
- Scheduling:
  - Unterstützung für GPUs, Cluster-Computer

Find out more about ProThOS on
https://manythreads.github.io/prothos/

or get in touch

Lutz Schubert
University of Ulm
lutz.schubert@uni-ulm.de