SIEMENS

Siemens Corporate Technology

Embedded Multicore Building Blocks (EMB²)

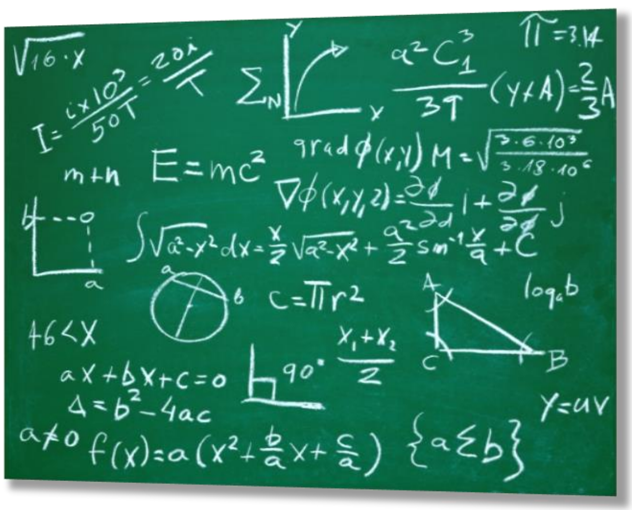
Dr. Tobias Schüle

Multicore Computing

Introduction
Sequential programming is easy (sometimes) …

Dot product (sequential)

```c
#define SIZE 1000

main() {
  double a[SIZE], b[SIZE];
  // Compute a and b ...
  double sum = 0.0;
  for(int i = 0; i < SIZE; i++)
    sum += a[i] * b[i];
  // Use sum ...
}
```



Introduction

… but multithreaded programming is tedious!

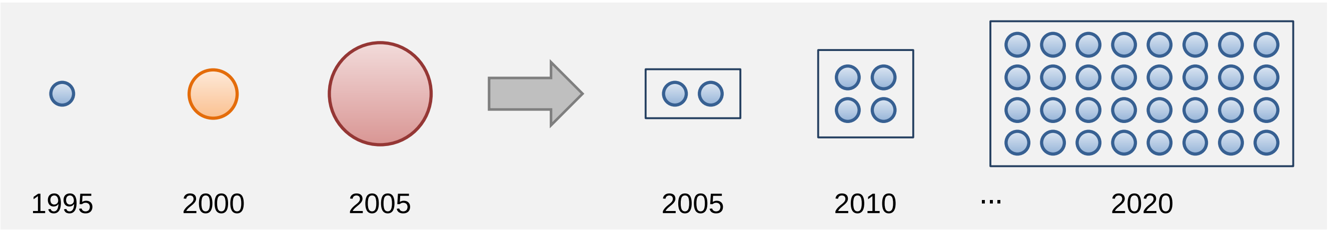
Dot product (POSIX threads)

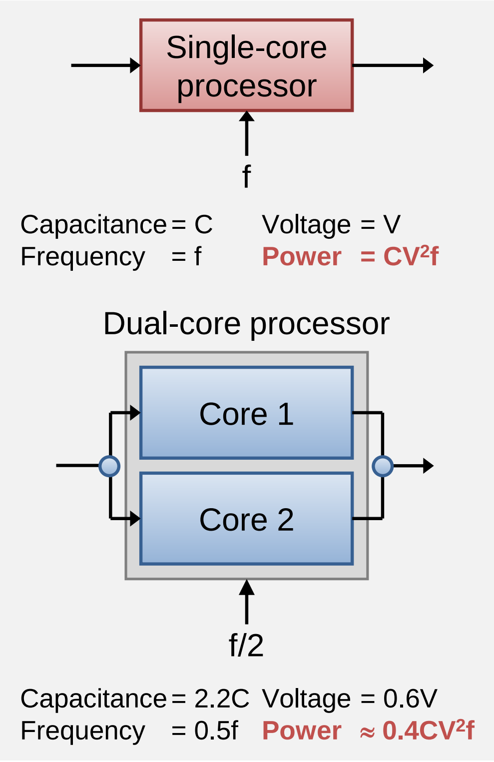
```
#include <iostream>
#include <pthread.h>

#define THREADS 4
#define SIZE 1000

using namespace std;

double a[SIZE], b[SIZE], sum;

pthread_mutex_t mutex_sum;

void *dotprod(void *arg) {
  int my_id = (int)arg;
  int my_first = my_id * SIZE/THREADS;
  int my_last = (my_id + 1) * SIZE/THREADS;

  double partial_sum = 0;
  for(int i = my_first; i < my_last && i < SIZE; i++)
    partial_sum += a[i] * b[i];

  pthread_mutex_lock(&mutex_sum);
  sum += partial_sum;
  pthread_mutex_unlock(&mutex_sum);

  pthread_exit((void*)0);
}
```

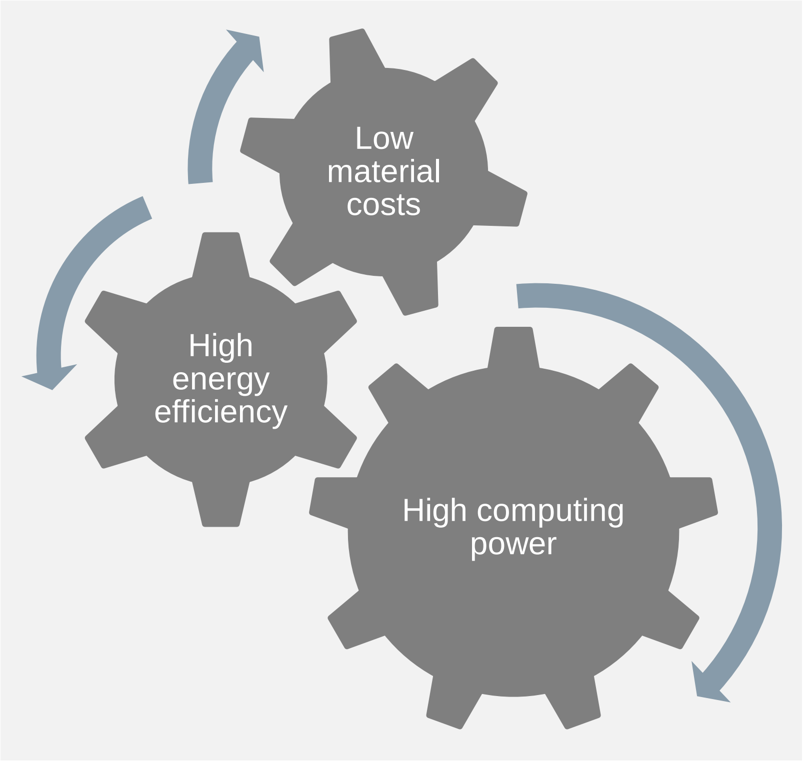
```
int main(int argc, char *argv[]) {
  // Compute a and b ...

  pthread_attr_t attr;
  pthread_t threads[THREADS];

  pthread_mutex_init(&mutex_sum, NULL);
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr,
    PTHREAD_CREATE_JOINABLE);

  sum = 0;
  for(int i = 0; i < THREADS; i++)
    pthread_create(&threads[i], &attr, dotprod,
                   (void*)i);

  pthread_attr_destroy(&attr);

  int status;
  for(int i = 0; i < THREADS; i++)
    pthread_join(threads[i], (void**)&status);

  // Use sum ...

  pthread_mutex_destroy(&mutex_sum);
  pthread_exit(NULL);
}
```

Barbara Chapman, Gabriele Jost, Ruud van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. MIT Press, 2007.

Introduction
Multicore processors are here to stay







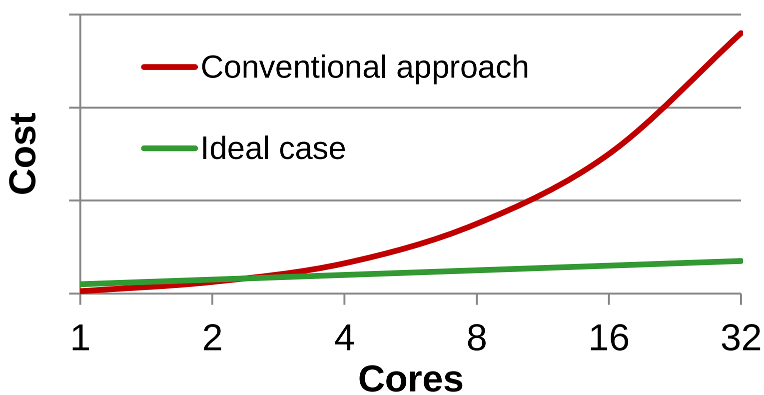
Source: Vishwani D. Agrawal

Introduction
"In 2022, multicore will be everywhere."

"Multicore has attracted wide attention from the **embedded systems community** […].

However, to obtain good multicore performance, **software is key for decomposing an original sequential program into parallel program parts** and assigning them to processor cores.

So far, such parallelization has been performed by application programmers, but it is **very difficult, takes a long time, and has a high cost**."



H. Alkhatib, P. Faraboschi, E. Frachtenberg, H. Kasahara, D. Lange, P. Laplante, A. Merchant, D. Milojicic, and K. Schwan. *IEEE CS 2022 Report*. IEEE Computer Society, 2014.
www.computer.org/cms/Computer.org/ComputingNow/2022Report.pdf

Introduction
Frameworks and Libraries for Parallel Programming

SIEMENS

OpenMP

Microsoft

Parallel Patterns Library (PPL)

intel

Threading Building Blocks (TBB)

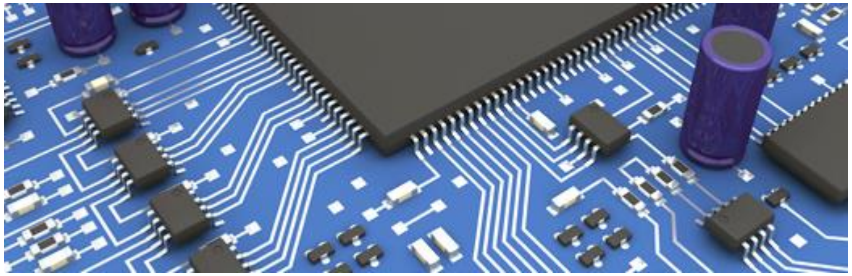
Apple's
Grand Central Dispatch

⋮

Most frameworks for parallel programming are intended for desktop/server applications and are **not suitable for embedded systems.**

Top challenges for multicore (IEEE CS 2022)

- Low-power scalable **homogeneous and heterogeneous architectures**
- **Hard real-time architectures** with local memory and their programming
- …



Embedded Multicore Building Blocks
Overview

SIEMENS

Embedded Multicore Building Blocks (EMB^2)

Domain-independent C/C++ library and runtime platform for embedded multicore systems.

Key features:

- Easy parallelization of existing code
- Resource-awareness (memory consumption)
- Real-time capability
- Fine-grained control over core usage (priorities, affinities)
- Support for distributed / heterogeneous systems
- Independence of hardware architecture (x86, ARM, …)

Embedded Multicore Building Blocks
Multicore Task Management API (MTAPI)

SIEMENS

MTAPI in a nut shell

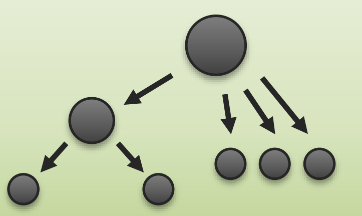
- **Standardized API** for task-parallel programming on a wide range of hardware architectures
- Developed and driven by practitioners of **market-leading companies**
- Part of Multicore-Association's **ecosystem** (MRAPI, MCAPI, …)

The Multicore Association

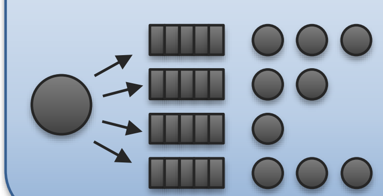
Contributing members:

SIEMENS
Working group lead
UH
Texas Instruments
Qualcomm
ENEA
PolyCore Software
freescale semiconductor
LSI
Plurality
Wind River

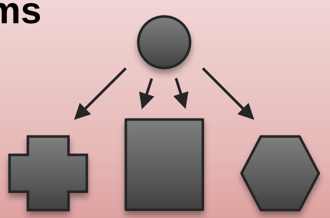
Tasks



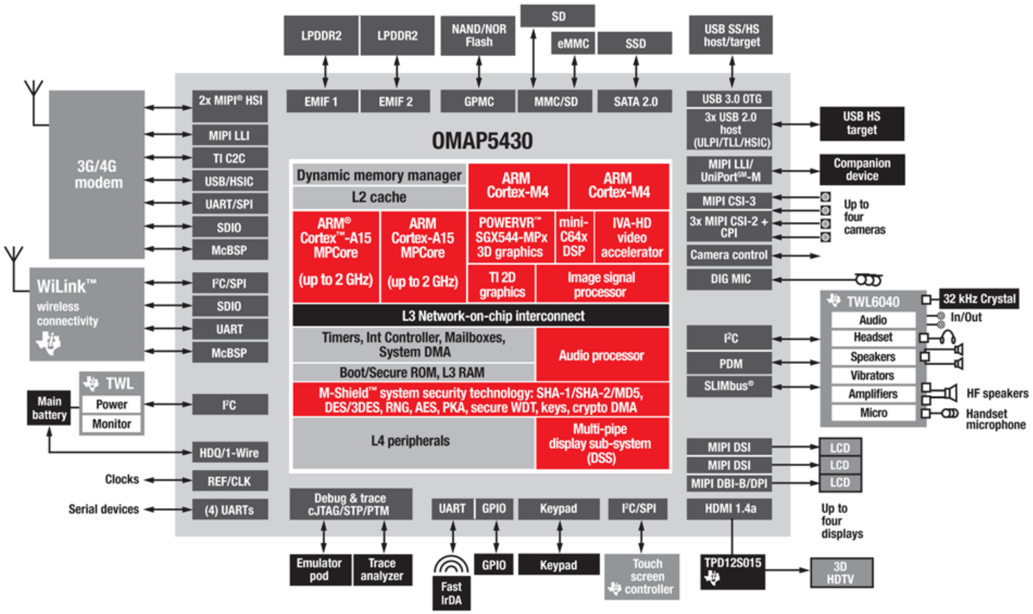
Queues



Heterogeneous Systems
- Shared memory
- Distributed memory
- Different instruction set architectures

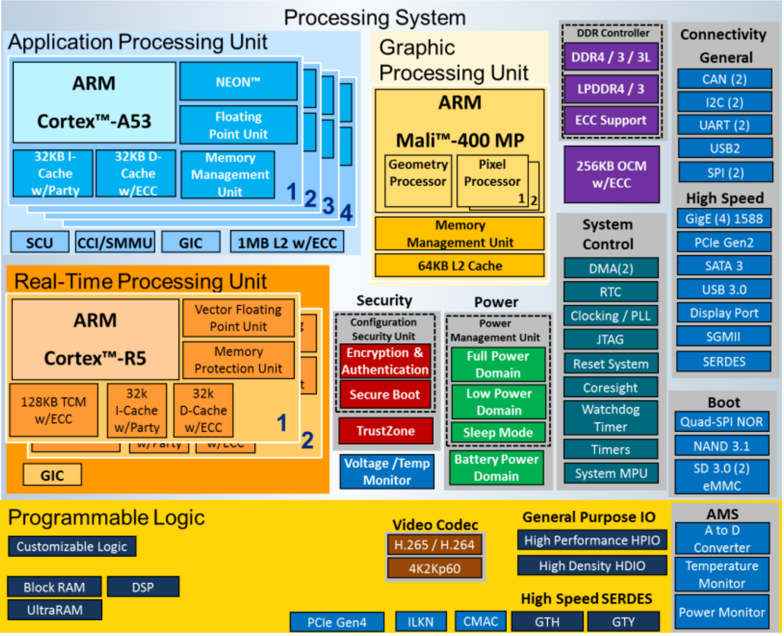


Embedded Multicore Building Blocks
Heterogeneous systems



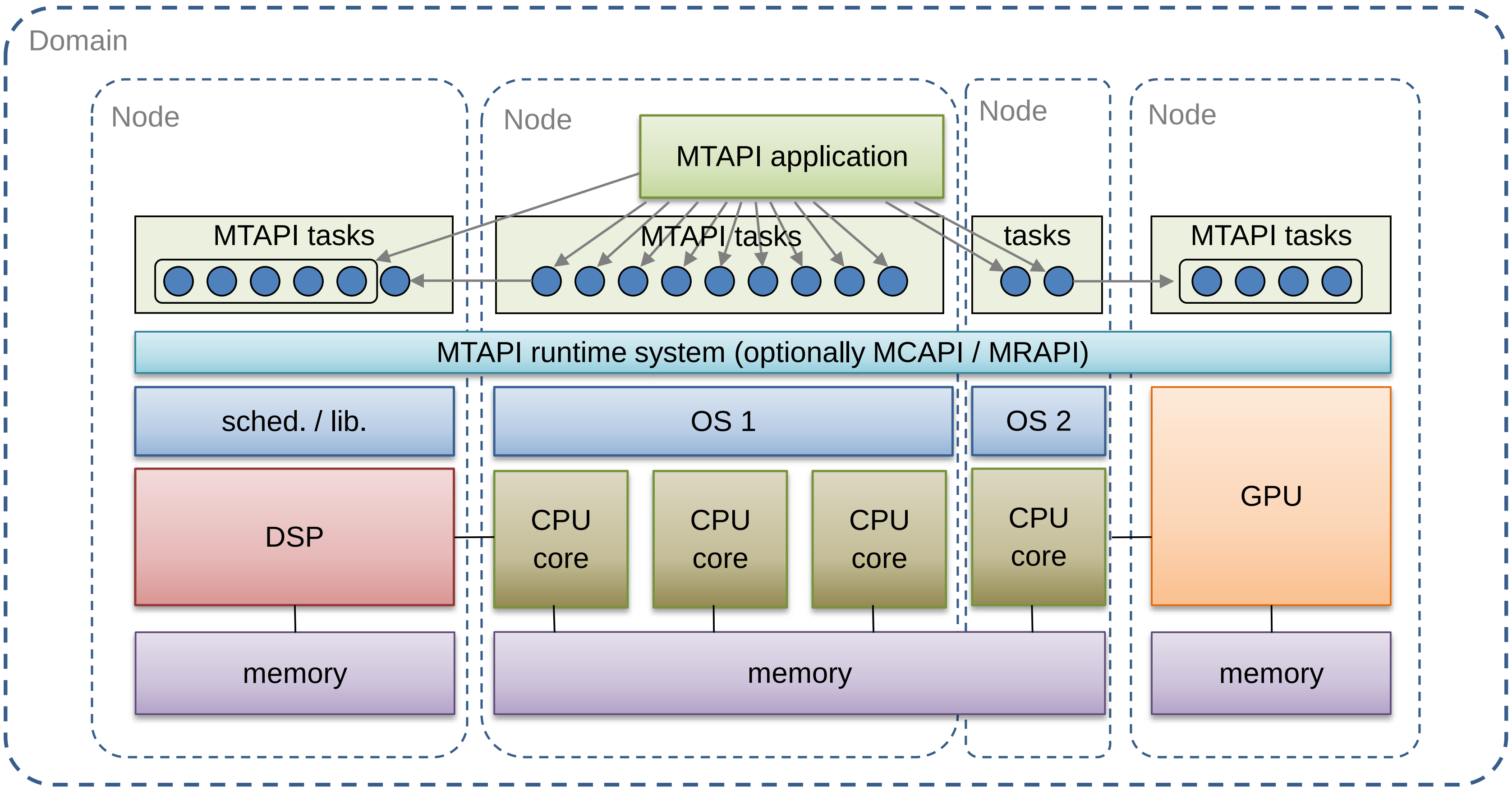
TI OMAP5430

Xilinx Zynq UltraScale MPSoC



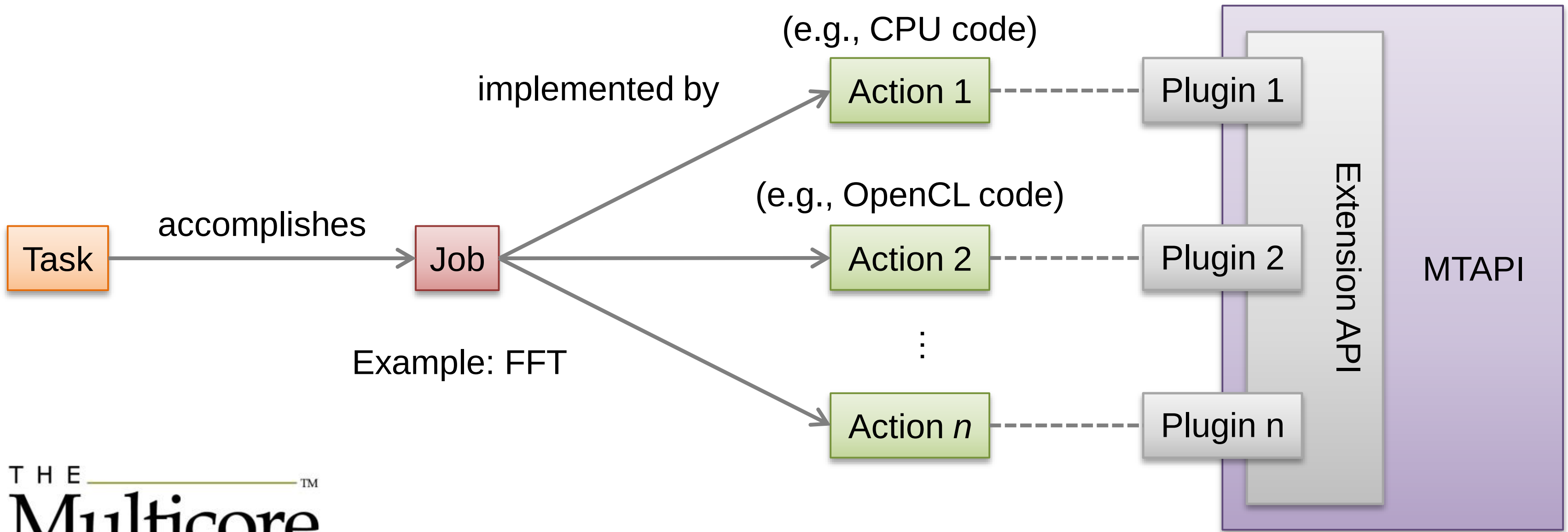
Corporate Technology

Embedded Multicore Building Blocks
MTAPI for Heterogeneous Systems (1)

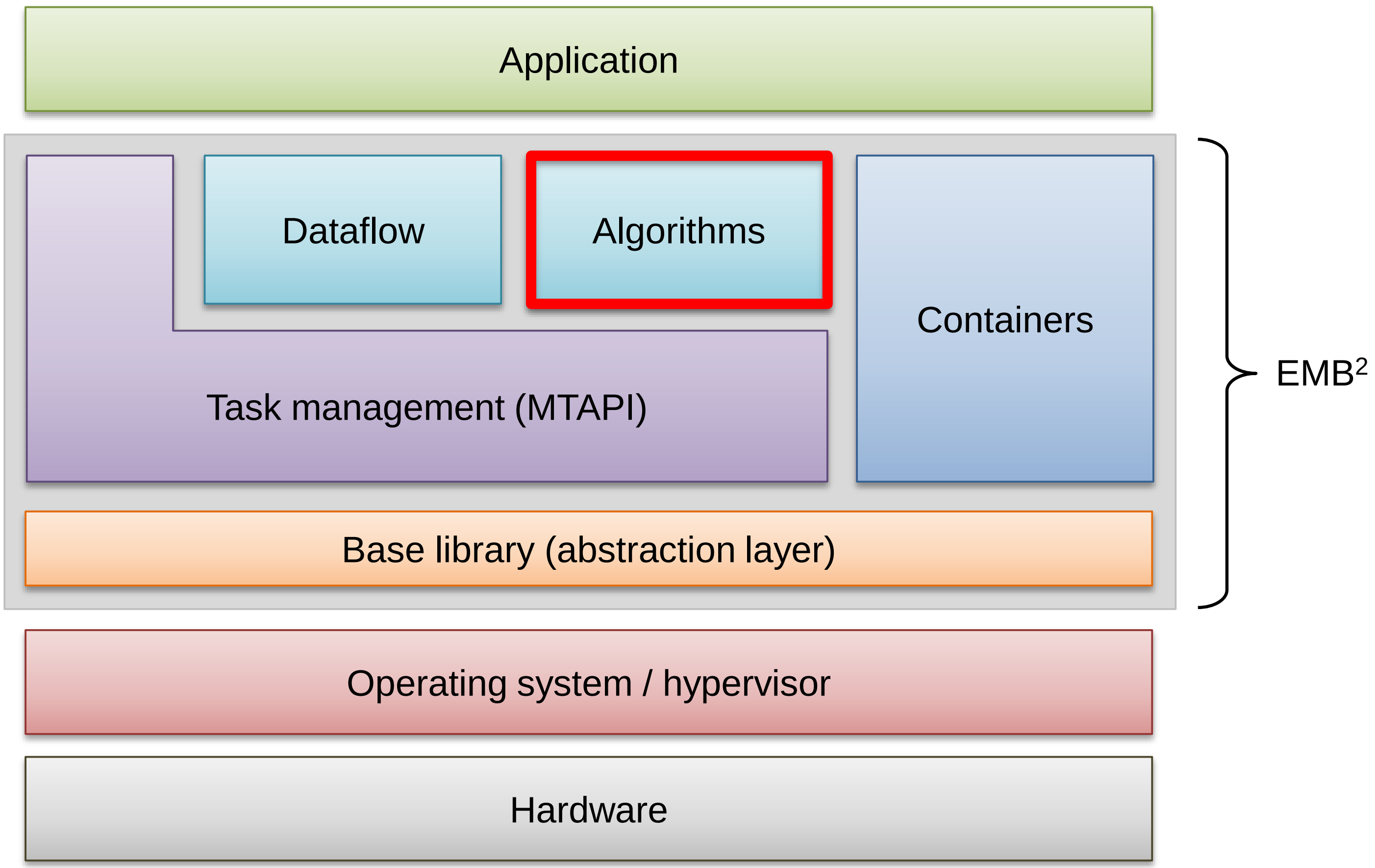


MTAPI for Heterogeneous Systems (2)

- **Job**: A piece of processing implemented by an action. Each job has a unique identifier.
- **Action**: Implementation of a job, may be hardware or software-defined.
- **Task**: Execution of a job resulting in the invocation of an action implementing the job associated with some data to be processed.



Embedded Multicore Building Blocks
Components



 Corporate Technology

Embedded Multicore Building Blocks
Algorithms and Task Affinities / Priorities

SIEMENS

Parallel for-each loop

```
std::vector<int> v;
// initialize v ...
embb::algorithms::ForEach(v.begin(), v.end(),
  [] (int& x) {x *= 2;}
);
```

No need to care of

- task creation and management
- number of processor cores
- load balancing and scheduling
- …

Function invocation

```
// Create execution policy
ExecutionPolicy policy(true, 0);
// Remove worker thread 0 fromaffinity set
policy.RemoveWorker(0);
// Start high priority tasks in parallel on
// specified worker threads (cores)
Invoke([=](){HighPrioFun1();},
       [=](){HighPrioFun2();},
       policy);
```

1st argument: affinity set (true = all)
2nd argument: priority (0 = highest)

Example: worker thread (core) 0 is reserved for special tasks

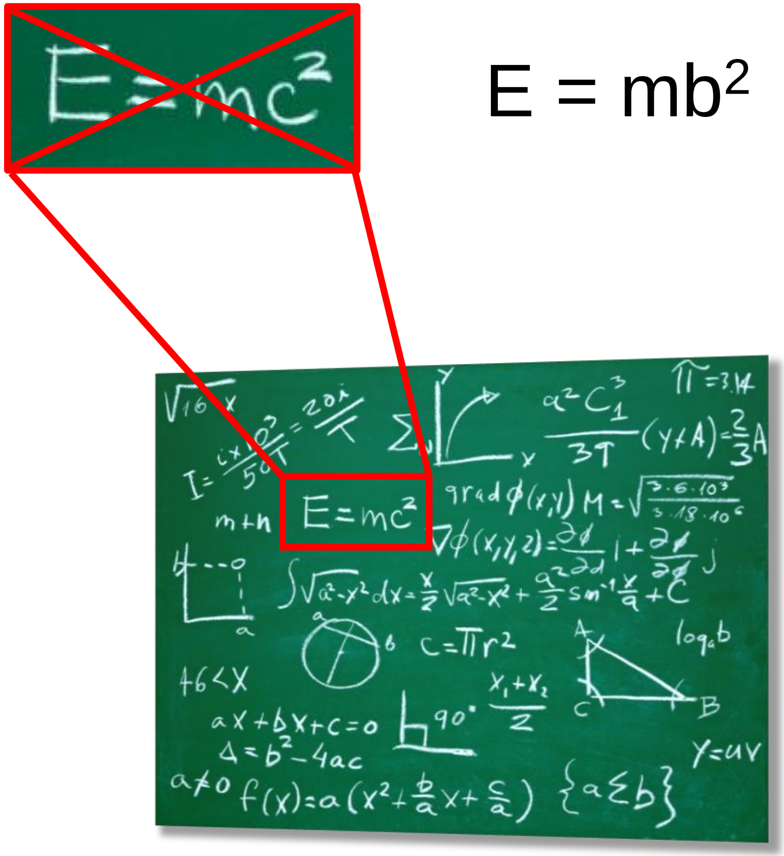
Pass policy as optional parameter

Embedded Multicore Building Blocks

Dot Product Revisited (1)

Dot product (sequential)

```
#define SIZE 1000

main() {
  double a[SIZE], b[SIZE];
  // Compute a and b ...
  double sum = 0.0;
  for(int i = 0; i < SIZE; i++)
    sum += a[i] * b[i];
  // Use sum ...
}
```



$E = mb^2$

Dot Product Revisited (2)

SIEMENS

Dot product (EMB^2)

```
#define SIZE 1000

main() {
  double a[SIZE], b[SIZE];
  // Compute a and b ...
  double sum = Reduce(
    Zip(&a[0], &b[0]), Zip(&a[SIZE], &b[SIZE]),
    0.0,
    std::plus<double>(),
    [] (const ZipPair<double&, double&>& p) {
      return p.First() * p.Second();
    }
  );
  // Use sum ...
}
```

Recipe (parallel algorithm)

Ingredients:

1. Input sequence
2. Neutral element
3. Reduction op.
4. Transformation fn.

No need to care of

- task creation and management
- number of processor cores
- load balancing and scheduling
- …

Task Affinities and Priorities

SIEMENS

Function invocation

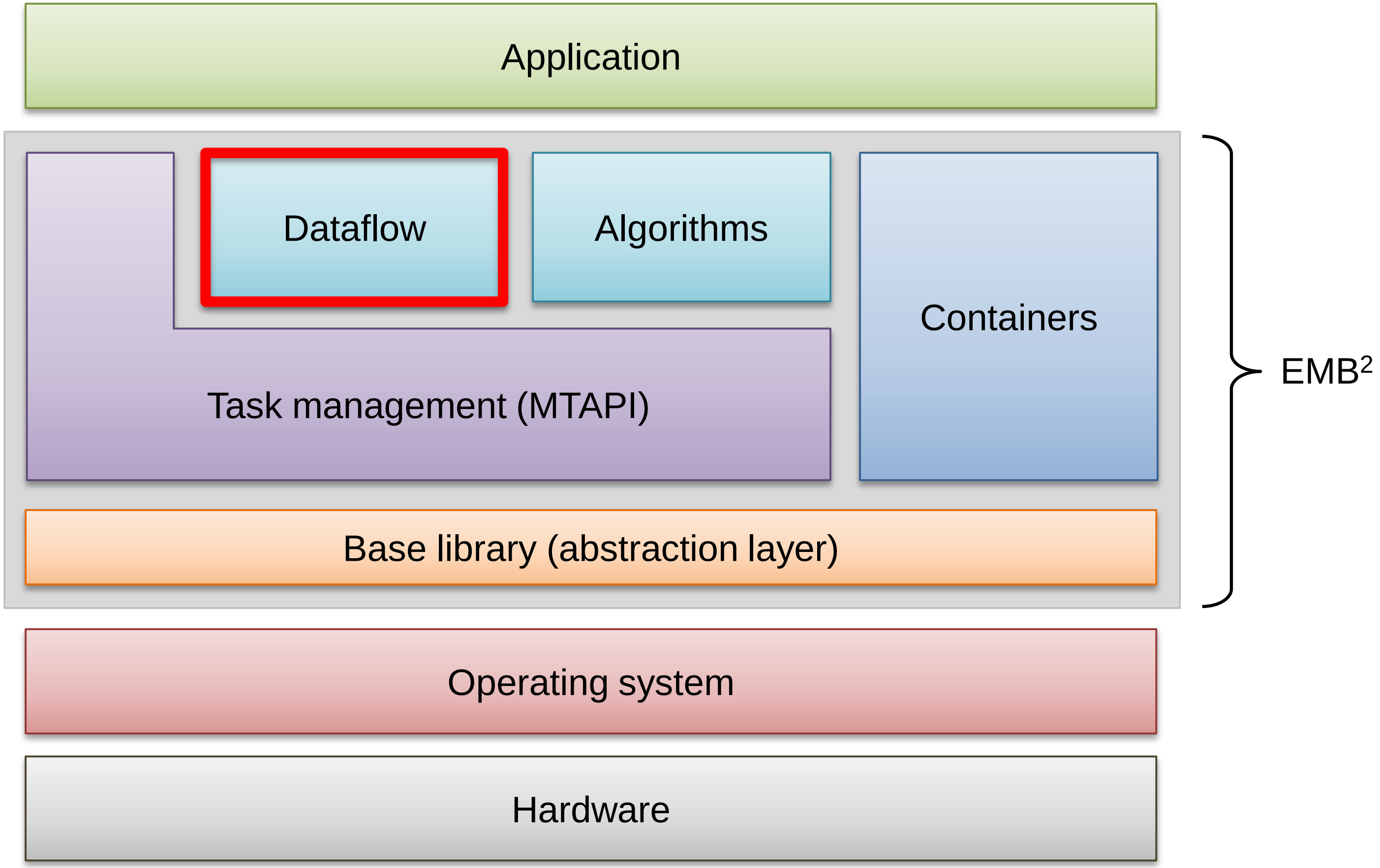
```
// Create execution policy
ExecutionPolicy policy(true, 0);
// Remove worker thread (core) 0 from
// affinity set
policy.RemoveWorker(0);
// Start high priority tasks in parallel on
// specified worker threads (cores)
Invoke([=](){HighPrioFun1();},
       [=](){HighPrioFun2();},
       policy);
```

1st argument: affinity set (true = all)
2nd argument: priority (0 = highest)

Example: worker thread (core) 0 is reserved for special tasks

Pass policy as optional parameter

Embedded Multicore Building Blocks
Components

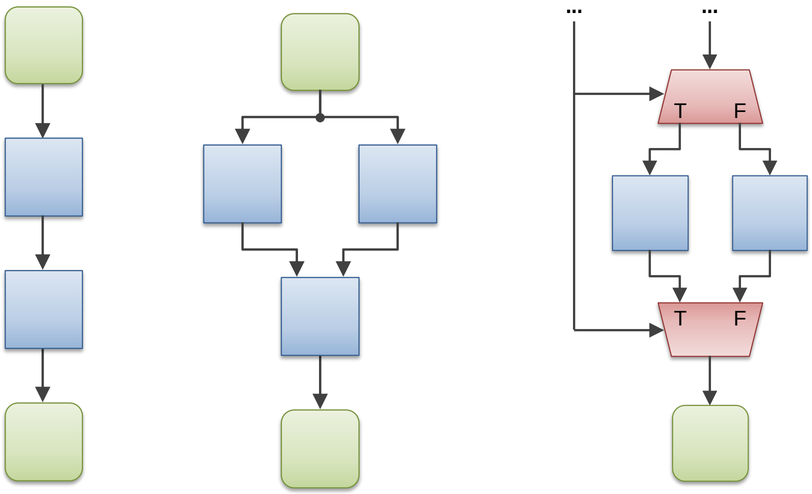


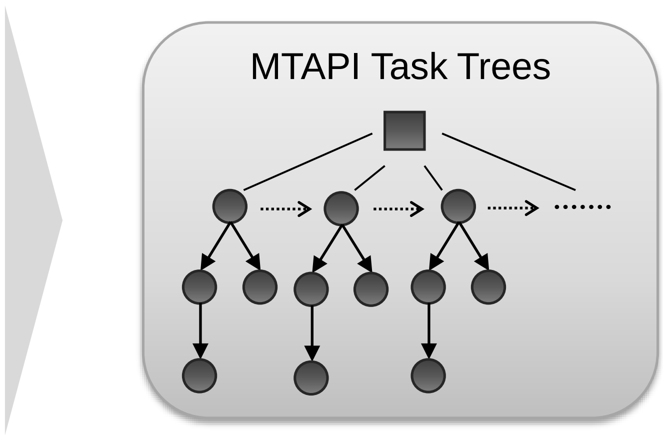
 Corporate Technology

Embedded Multicore Building Blocks
Dataflow Framework

Stream processing

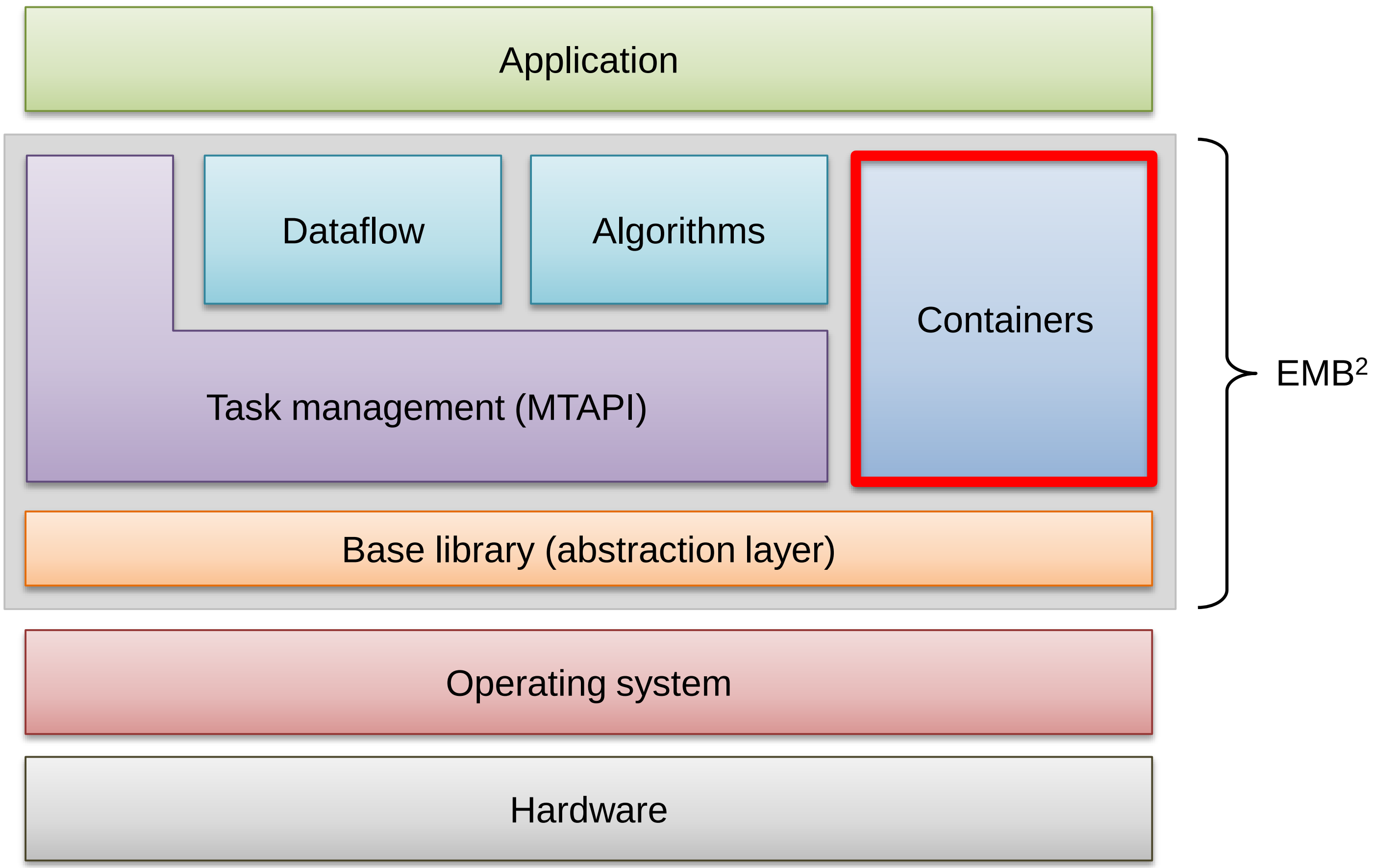
- Embedded systems frequently process **continuous streams of data** such as
  - sensor and actuator data,
  - network packets, …
  - medical images, …
- Such applications can be modeled using **dataflow networks** and executed in parallel





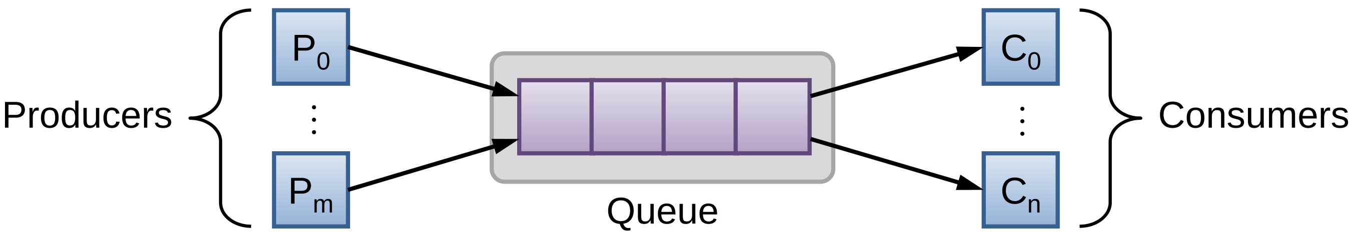
 Corporate Technology

Embedded Multicore Building Blocks
Components



Embedded Multicore Building Blocks
Container Requirements

1. **No race conditions** in case of concurrent accesses ⇨ **Thread safety**
2. **No unpredictable delays** in case of contention ⇨ **Progress guarantee**
3. **No dynamic memory allocation** after startup ⇨ **Preallocated memory**



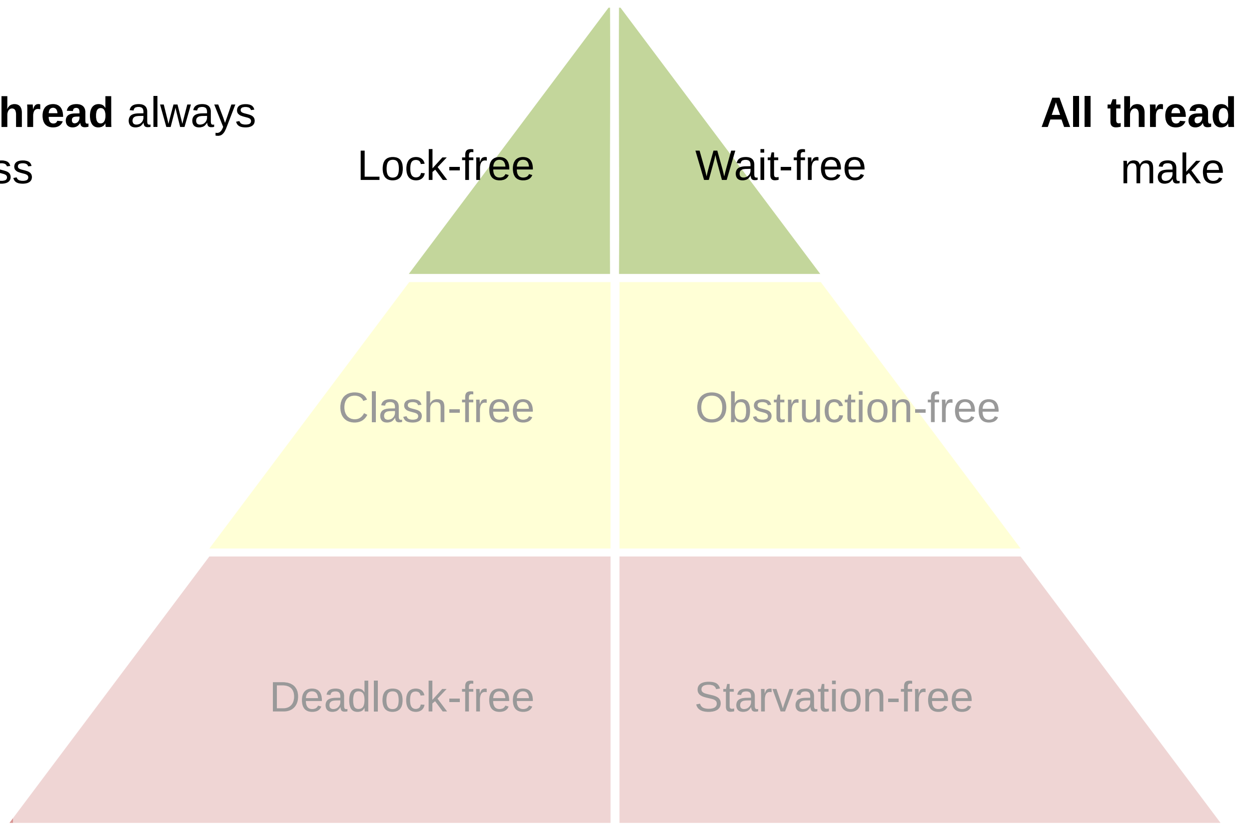
Implementation	Thread safety	Progress guarantee	Preallocated memory
`std::queue` `QQueue (Qt)`	✗	—	✗
`std::queue` `QQueue (Qt)` `+ Mutex`	✓	✗	✗
`boost::lockfree::queue` `tbb::concurrent_queue`	✓	✓ / ?	✗ / ?
`embb::LockFreeMPMCQueue` `embb::WaitFreeSPSCQueue`	✓	✓	✓

 Corporate Technology

Embedded Multicore Building Blocks
Progress Guarantees

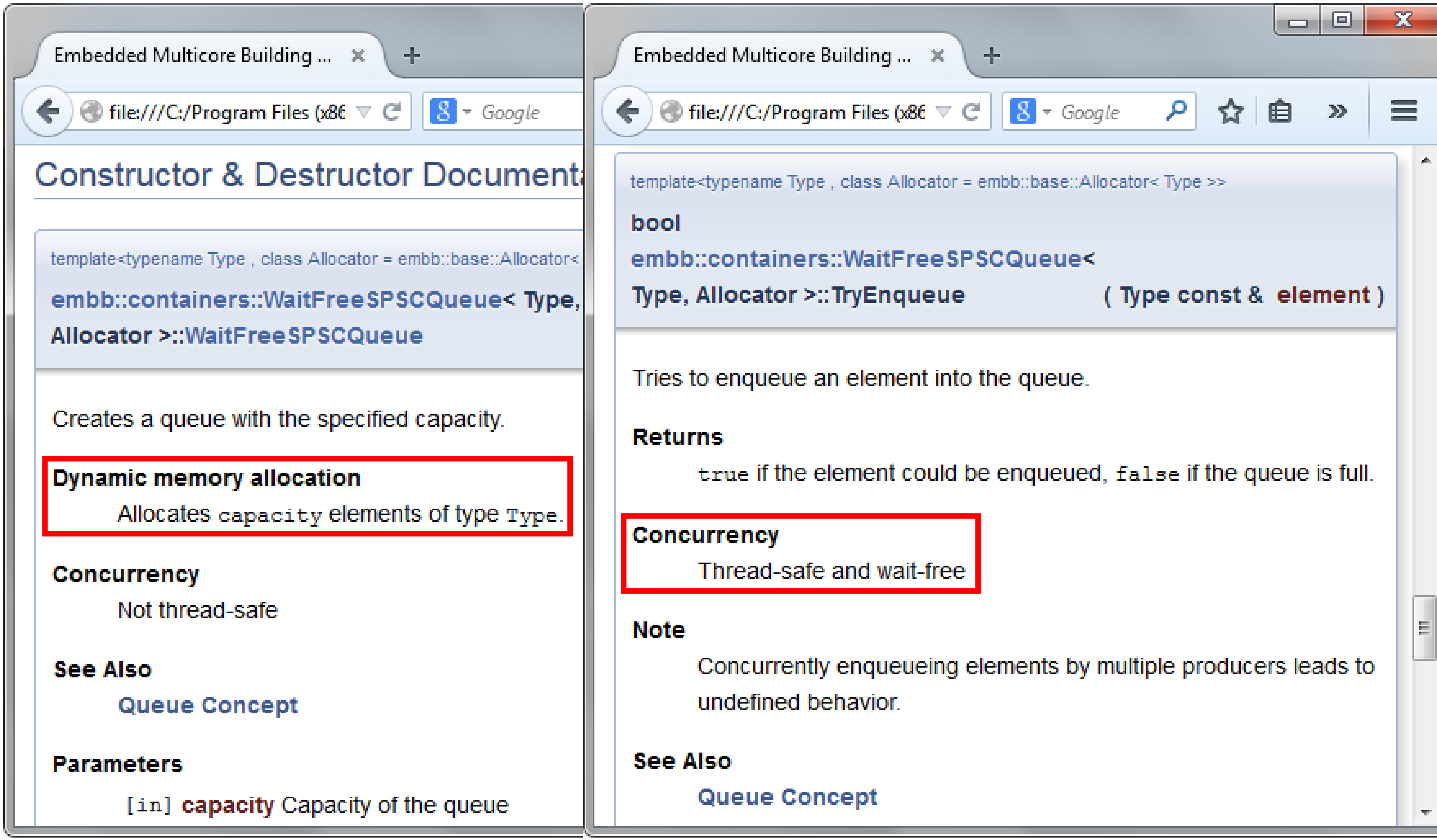
At least one thread always makes progress

All threads always make progress



M. Herlihy and N. Shavit. “On the nature of progress”. International conference on Principles of Distributed Systems (OPODIS'11), Springer, 2011.

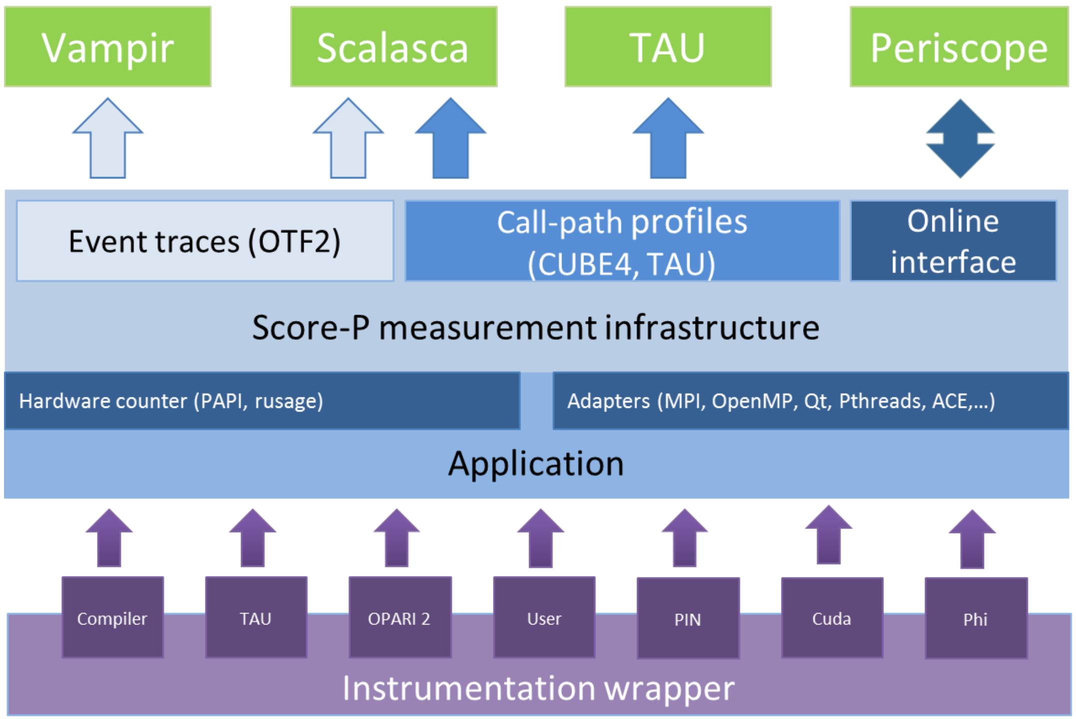
Embedded Multicore Building Blocks
Lock-free / Wait-free Algorithms



Properties of Lock-free / Wait-free Algorithms

- **Progress guarantees**
  With wait-freedom, the completion of an operation is guaranteed to occur in a finite number of steps. Lock-freedom guarantees the overall progress of a system.
- **Deadlock absence**
  Wait-free and lock-free data structures are immune to deadlock conditions.
- **Signal safety**
  Coherency in the context of asynchronous interruptions is guaranteed.
- **Termination safety**
  Linearizable operations may be aborted at any time without sacrificing the overall availability of a system.
- **Priority inversion avoidance**
  Wait-free algorithms cannot prevent high priority threads from making progress.

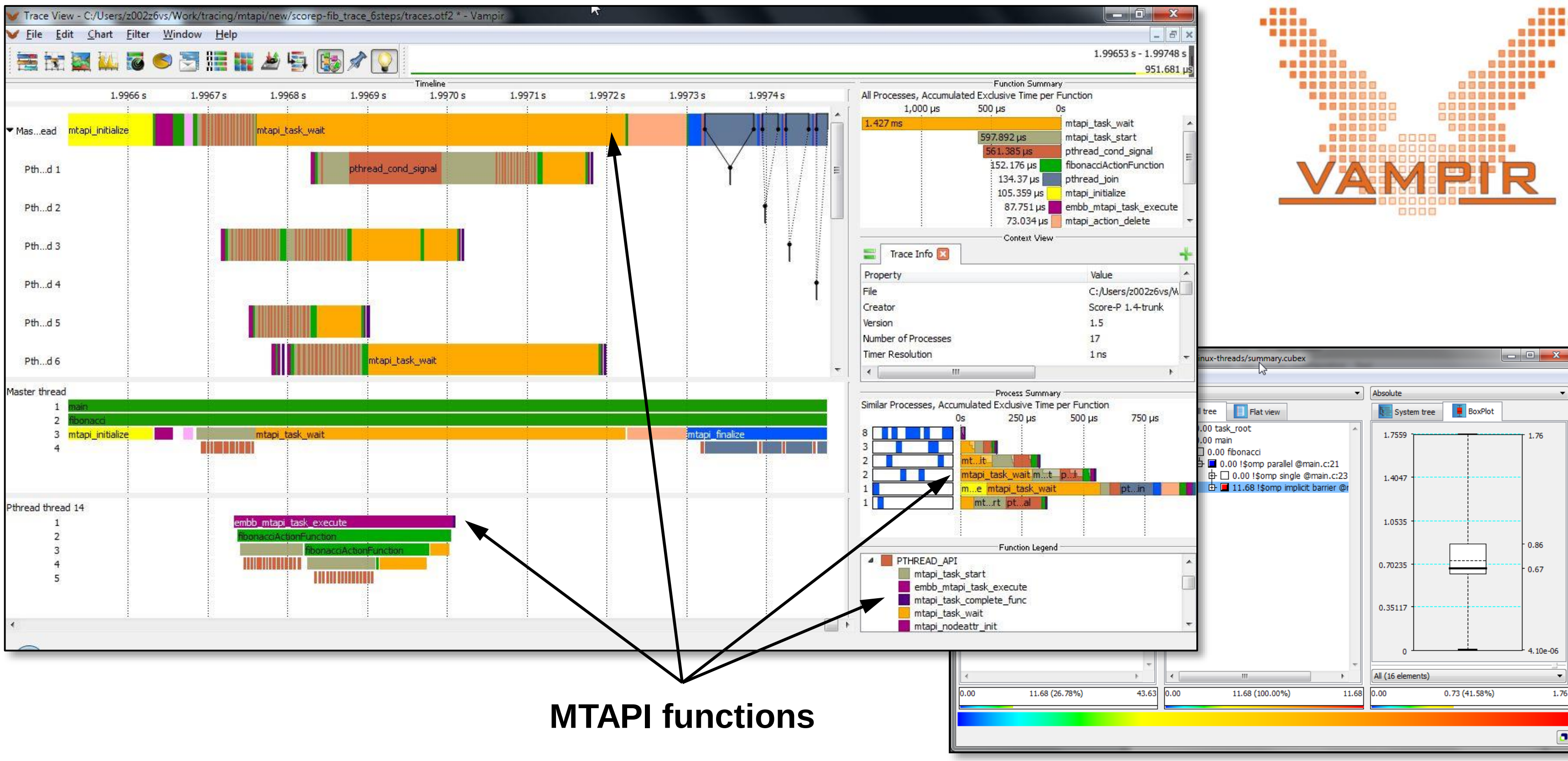
Tracing/Profiling MTAPI Applications with Score-P



- Open source community
- Linux & Windows
- Platform independent (x86, ARM, and PPC)
- Heterogeneous system support (e.g., Intel Phi, CUDA)
- Open formats enabling interoperability and custom analysis types
- Extremely scalable

www.score-p.org

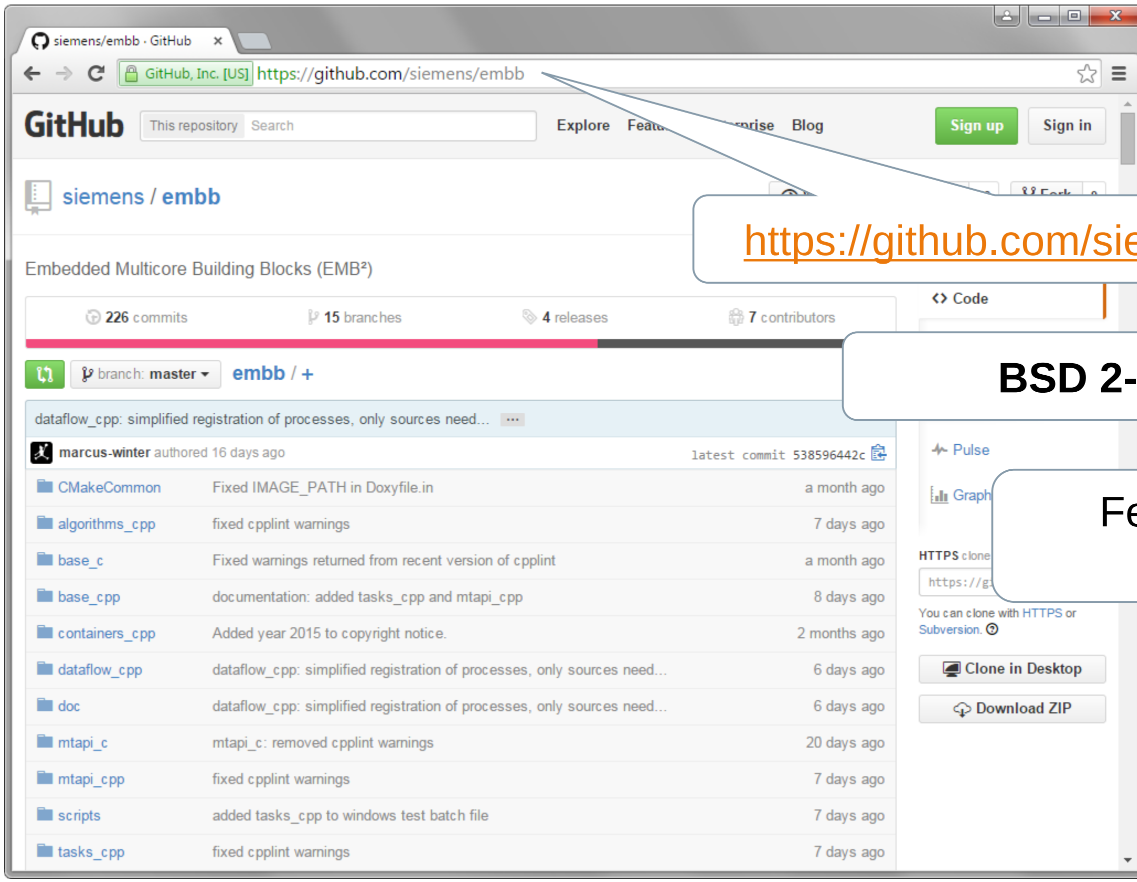
Embedded Multicore Building Blocks
Visualization and Interpretation



MTAPI functions

- Adaptation of well-established and widely used Score-P framework to MTAPI ⇨ Benefit from experience in HPC
- Cooperation with Jülich Supercomputing Centre (JSC)

Embedded Multicore Building Blocks
Hello World!



https://github.com/siemens/embb/

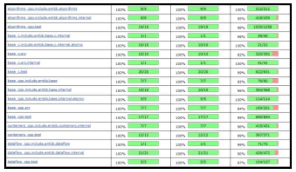
BSD 2-clause license

Feedback and contributions are very welcome!

Embedded Multicore Building Blocks
Code Quality

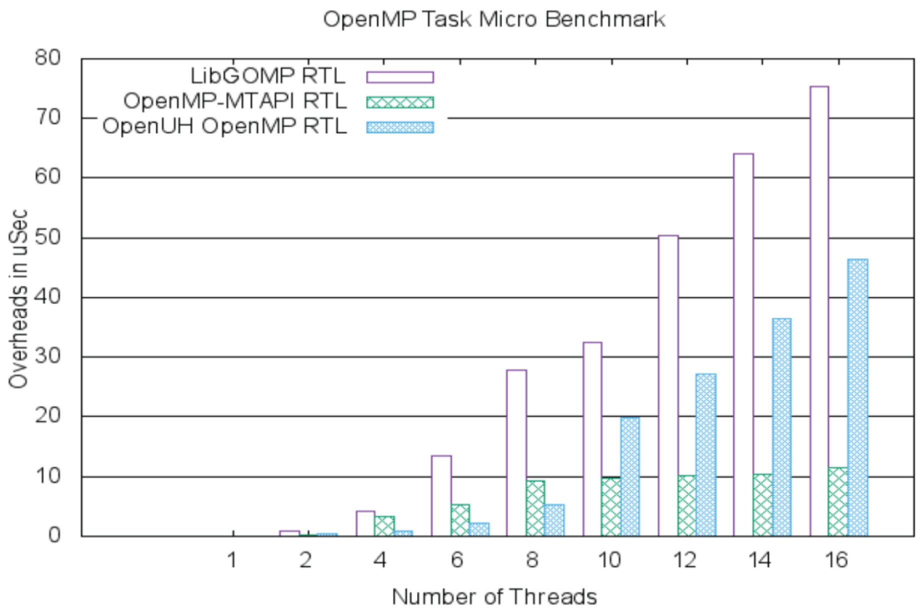
Agile development process

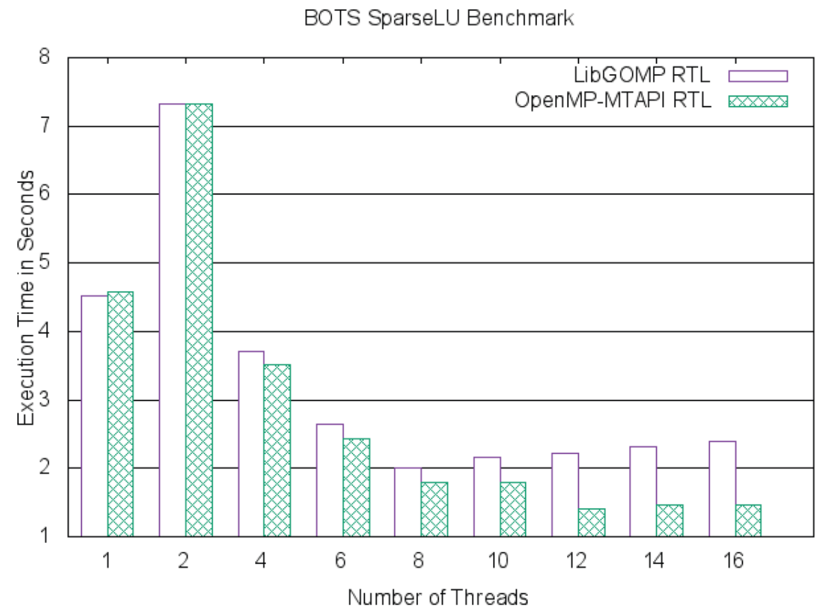
- Formal verification (partially)
- Static source code analysis
- Linearizability checker
- Rule checker (cppcheck)
- Continuous integration
- Unit tests (> 90% statement coverage)
- Workflow-driven design/code reviews
- Coding guidelines (Google’s cpplint)
- Zero compiler warnings



Performance Comparison

Measurements from University of Houston show efficiency of EMB² (green bars):



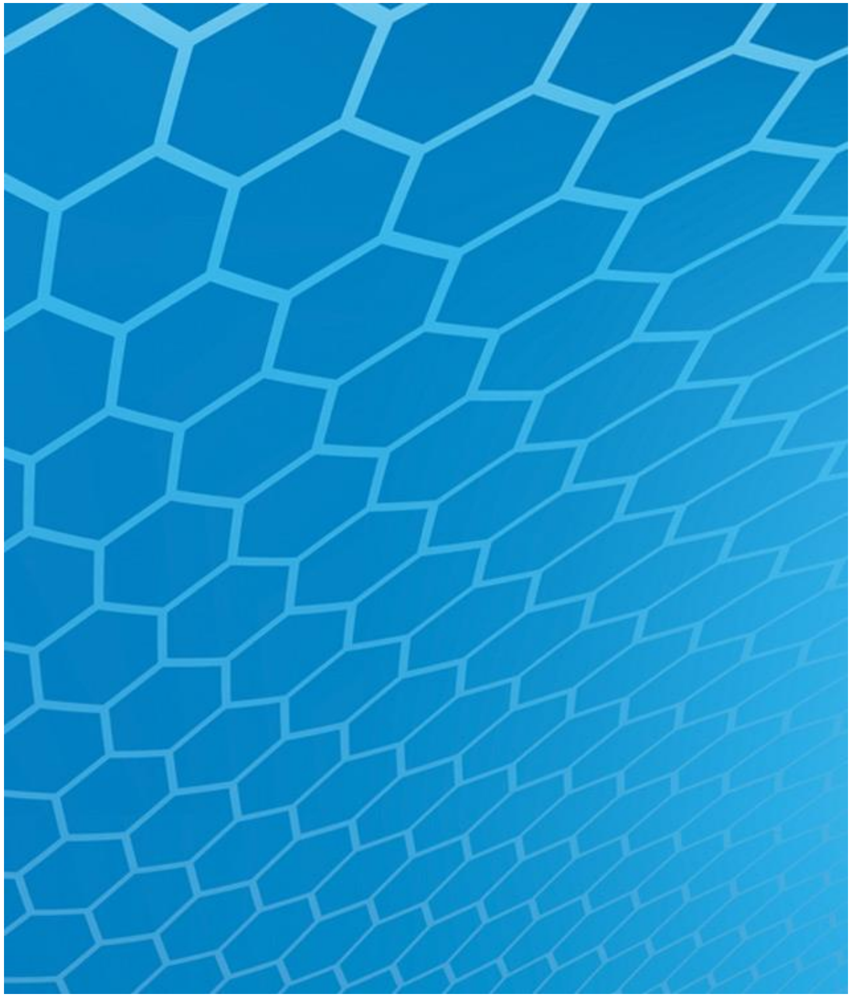


P. Sun, S. Chandrasekaran, S. Zhu, and B. Chapman. *Deploying OpenMP Task Parallelism on Multicore Embedded Systems with MCA Task APIs*. International Conference on High Performance Computing and Communications (HPCC), IEEE, 2016.

Embedded Multicore Building Blocks
Summary

- Efficient software development
- High performance and scalability
- Improved code quality (prevention of concurrency bugs)
- Suitable for embedded systems (memory and real-time constraints)

Contact



Dr. Tobias Schüle
Siemens AG
Corporate Technology
CT RDA ITP SES-DE

Otto-Hahn-Ring 6
81739 München
Germany

Email: tobias.schuele@siemens.com