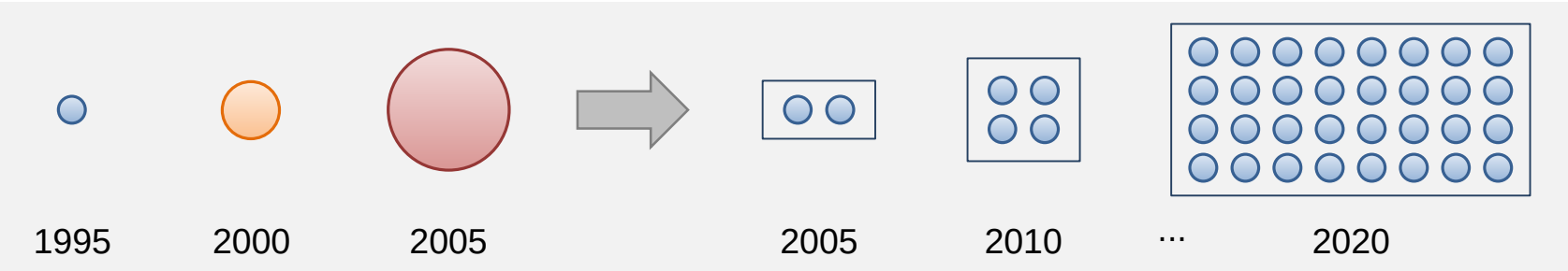
FOSDEM'16

Embedded Multicore Building Blocks (EMB²)

Easy and Efficient Parallel Programming of Embedded Systems

Tobias Schüle
Siemens Corporate Technology

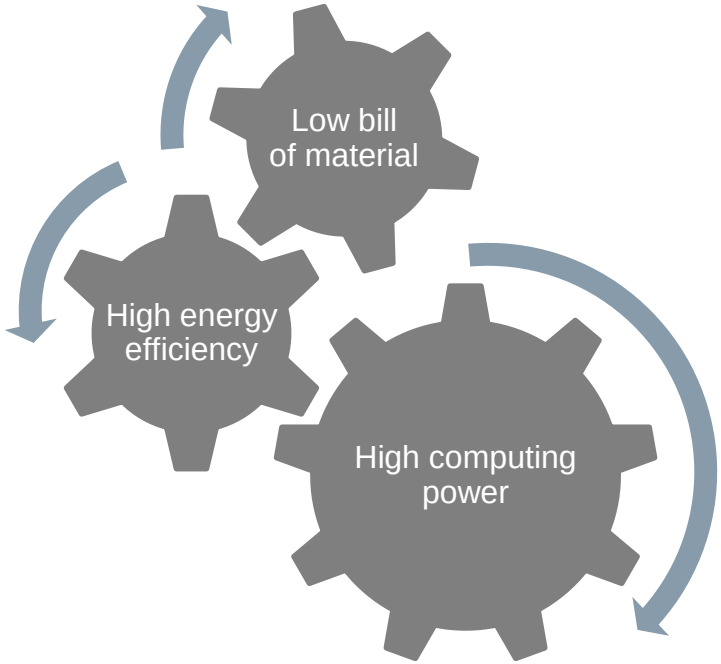
Introduction
“The free lunch is over!”



- **No further increase of clock frequencies** due to excessive heat dissipation
- **No significant performance improvements** of single processor cores

⇨ **Multicore processors are here to stay**

⇨ **Applications need to leverage parallelism**

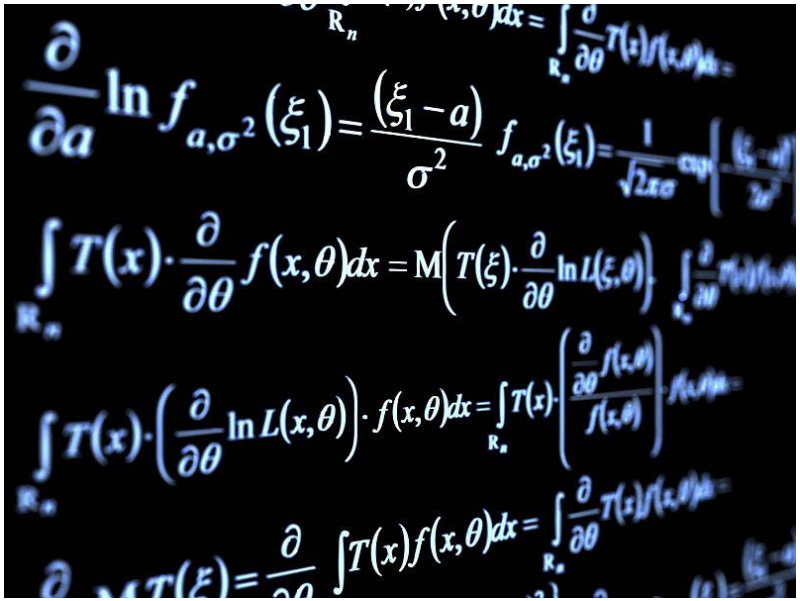


Introduction
Sequential programming is easy (sometimes) …

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 ...
}
```



http://wallpoper.com/wallpaper/formula-mathematics-255330
Public Domain

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
“In 2022, multicore will be everywhere.” (IEEE Computer Society)

Various libraries and language extensions for parallel programming available:

- OpenMP
- Intel’s Threading Building Blocks
- Apple’s Grand Central Dispatch
- …

} **Target desktop/server applications**
Not suitable for embedded systems

Top challenges for multicore (IEEE CS 2022 Report)[1]

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

[1] 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

“Multicore has attracted wide attention from the **embedded systems community** […]. So far, such parallelization has been performed by application programmers, but it is **very difficult, takes a long time, and has a high cost**.”

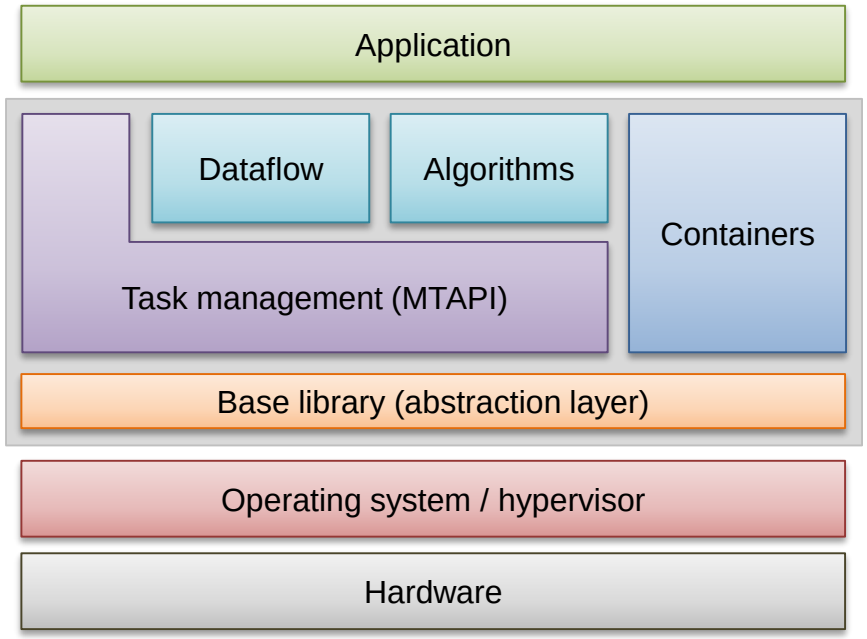
Embedded Multicore Building Blocks (EMB²)

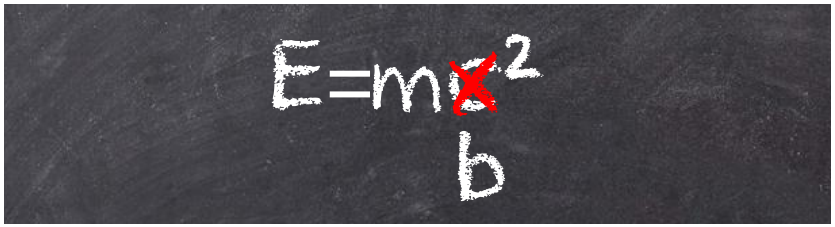
EMB² at a glance

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

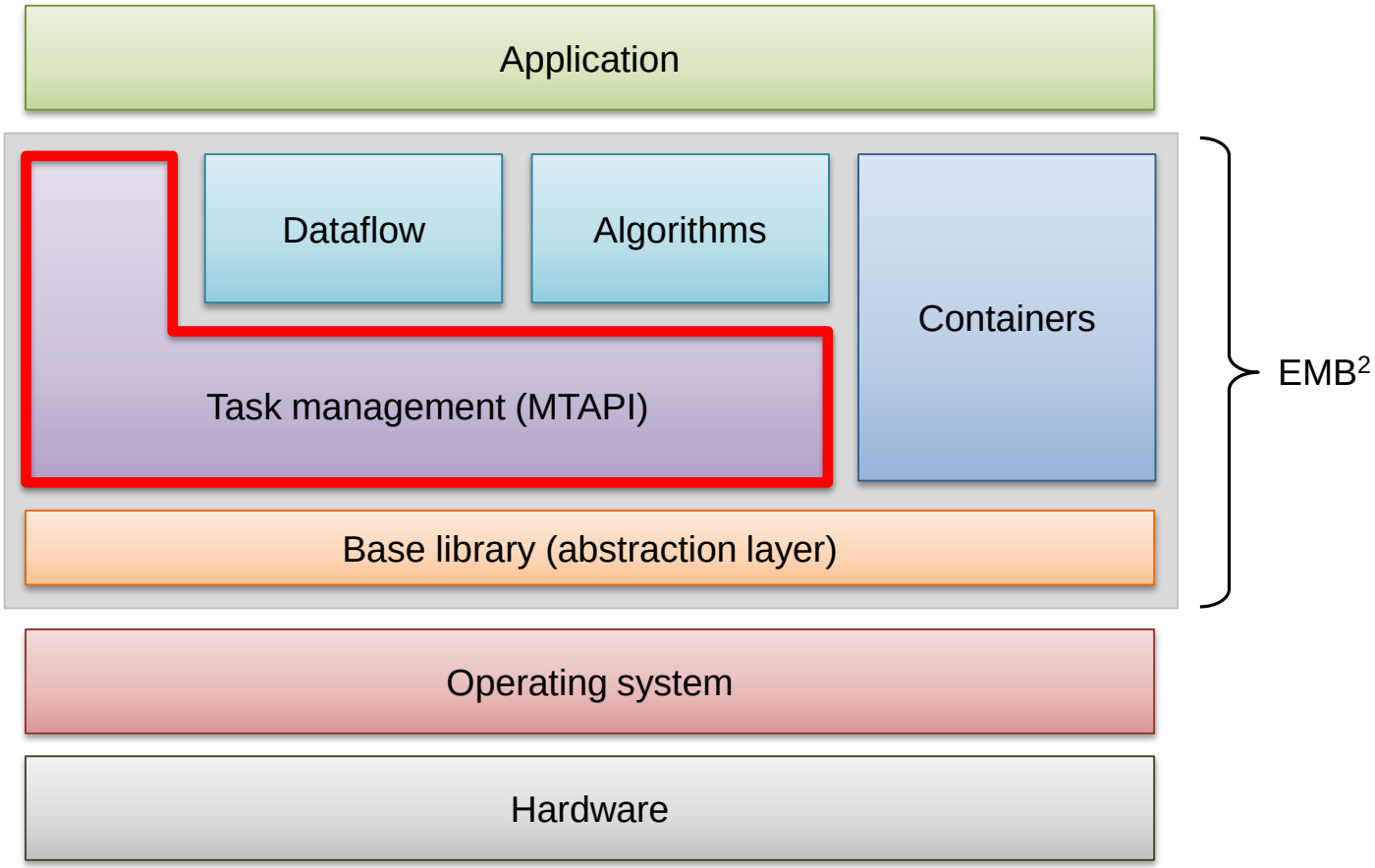
Key features:

- Easy parallelization of existing code
- Real-time capability, resource awareness
- Fine-grained control over core usage (task priorities, affinities)
- Lock-/wait-free implementation
- Support for heterogeneous systems
- Independence of hardware architecture (x86, ARM, …)





Embedded Multicore Building Blocks
Components



Embedded Multicore Building Blocks
Multicore Task Management API (MTAPI)

MTAPI in a nut shell

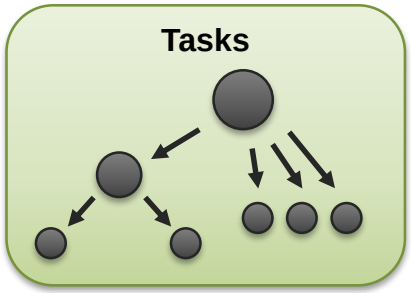
- **Standardized API** for task-parallel programming on a wide range of hardware architectures
- Developed and driven by practitioners of **market-leading companies**
- Part of Multicore Association's **ecosystem** (MRAPI, MCAPI, …)

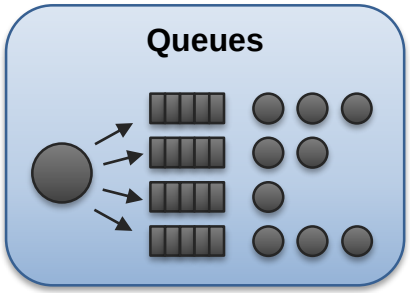
Contributing members:

THE Multicore ASSOCIATION

SIEMENS
Working group lead

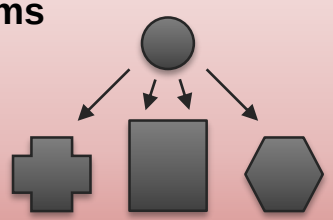
TEXAS INSTRUMENTS
QUALCOMM
ENEA
PolyCore Software
freescale semiconductor
LSI
PLURALITY
WIND RIVER





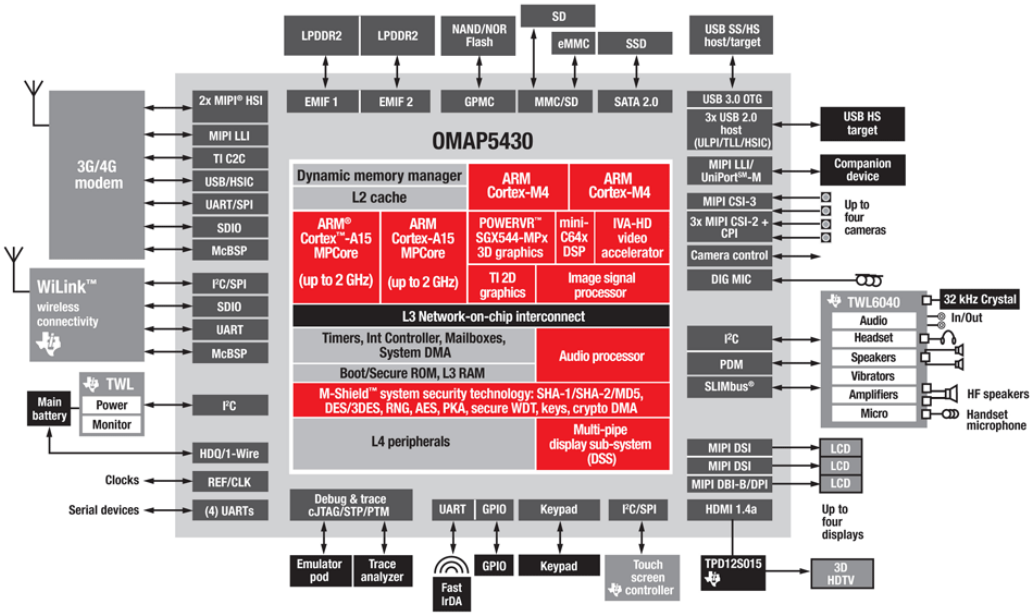
Heterogeneous Systems

- Shared memory
- Distributed memory
- Different instruction set architectures



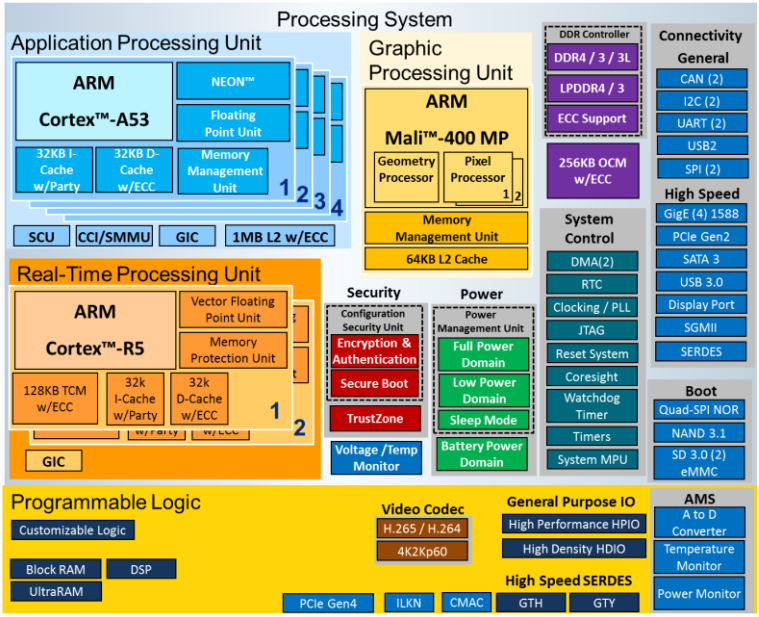
Embedded Multicore Building Blocks
Heterogeneous systems

TI OMAP5430



© Texas Instruments

Xilinx Zynq UltraScale MPSoC

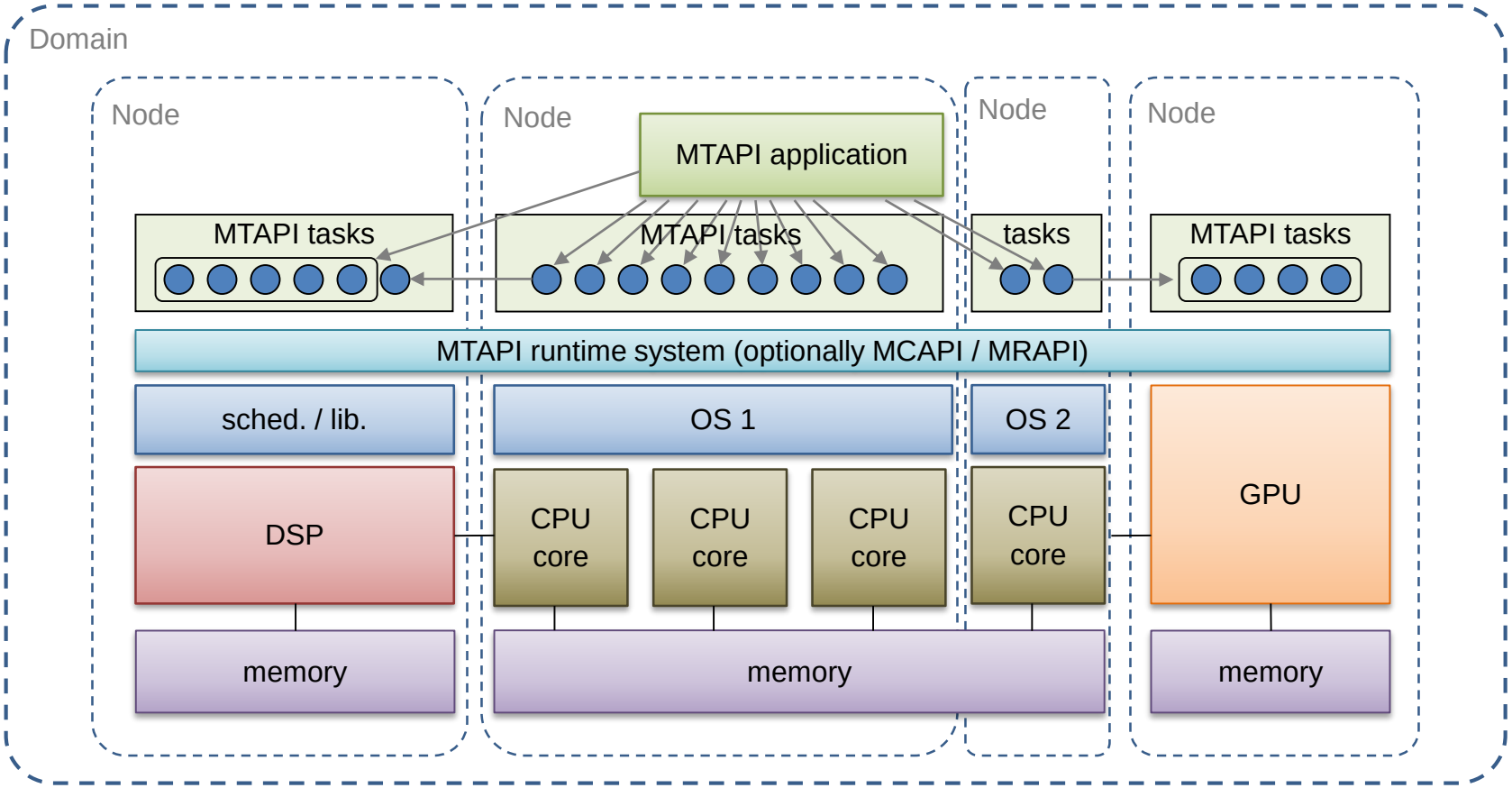


© Xilinx

Key characteristics

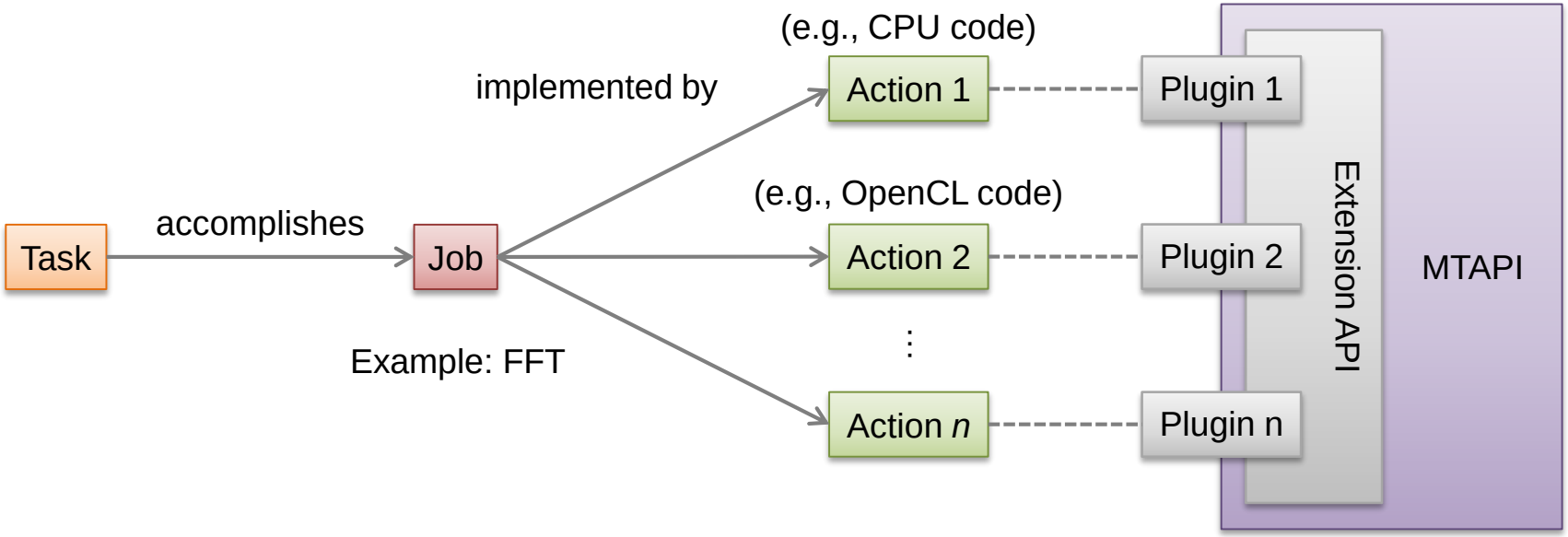
- High performance using specialized hardware (DSPs, FPGAs, etc.)
- Low power consumption ⇨ reduced heat dissipation
- Complex programming due to different architectures (lack of abstraction)

Embedded Multicore Building Blocks
MTAPI for Heterogeneous Systems (1)

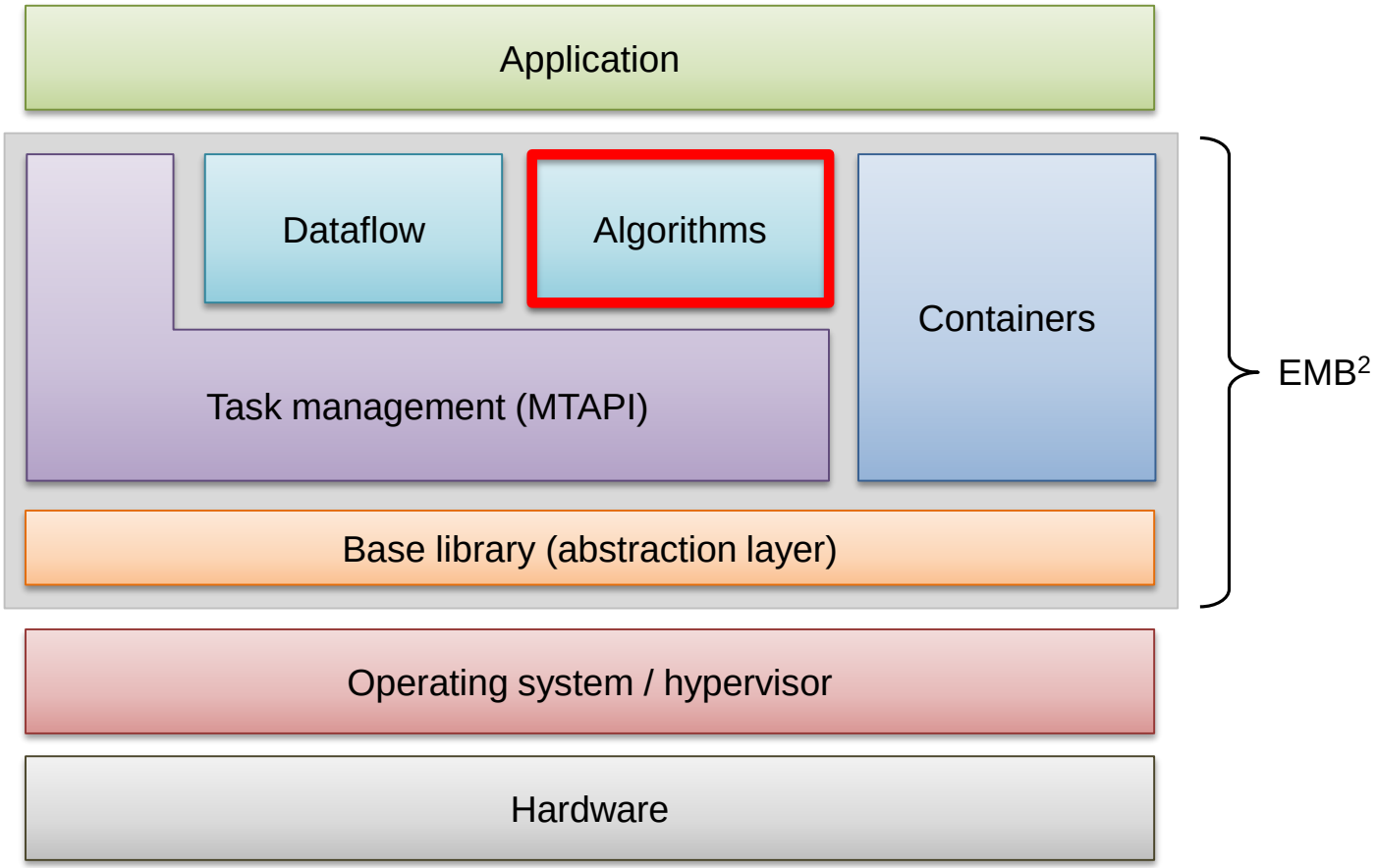


MTAPI for Heterogeneous Systems (2)

- **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



Embedded Multicore Building Blocks
Algorithms and Task Affinities / Priorities

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

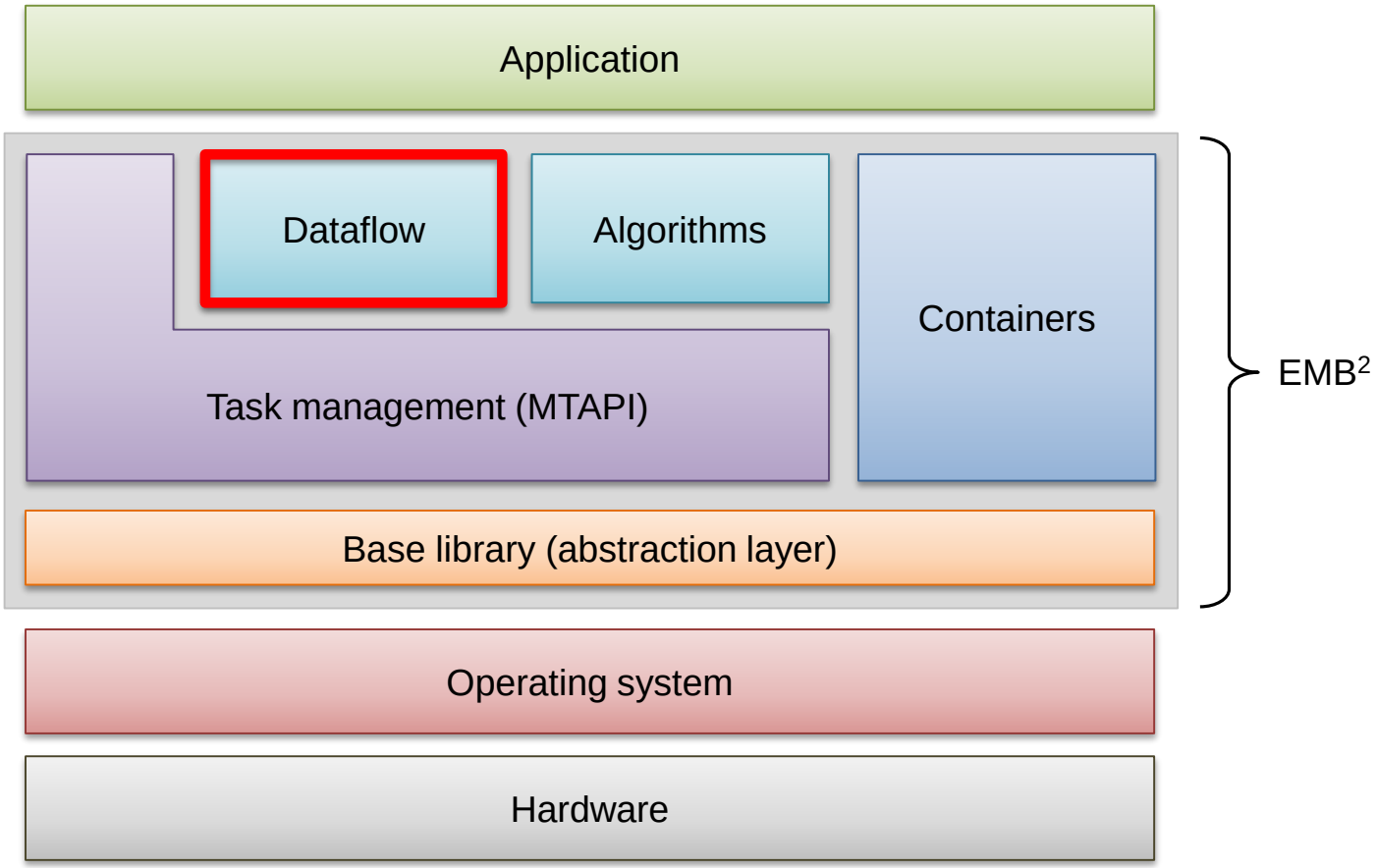
```
// Create execution policy
ExecutionPolicy policy(true, 0);
// Remove worker thread 0 fromaffinity set
policy.RemoveWorker(0);
// Start high priority tasks in parallel on
// specified worker threads (cores)
Invoke([=](){HighPrioFun1();},
       [=](){HighPrioFun2();},
       policy);
```

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

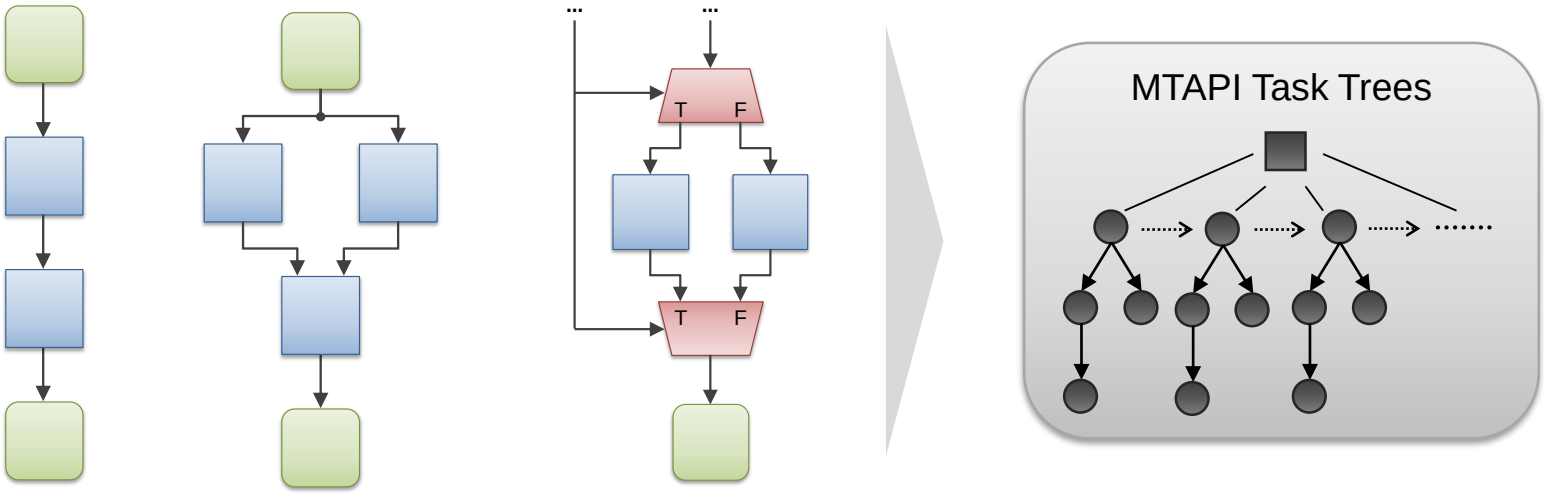
Embedded Multicore Building Blocks
Components



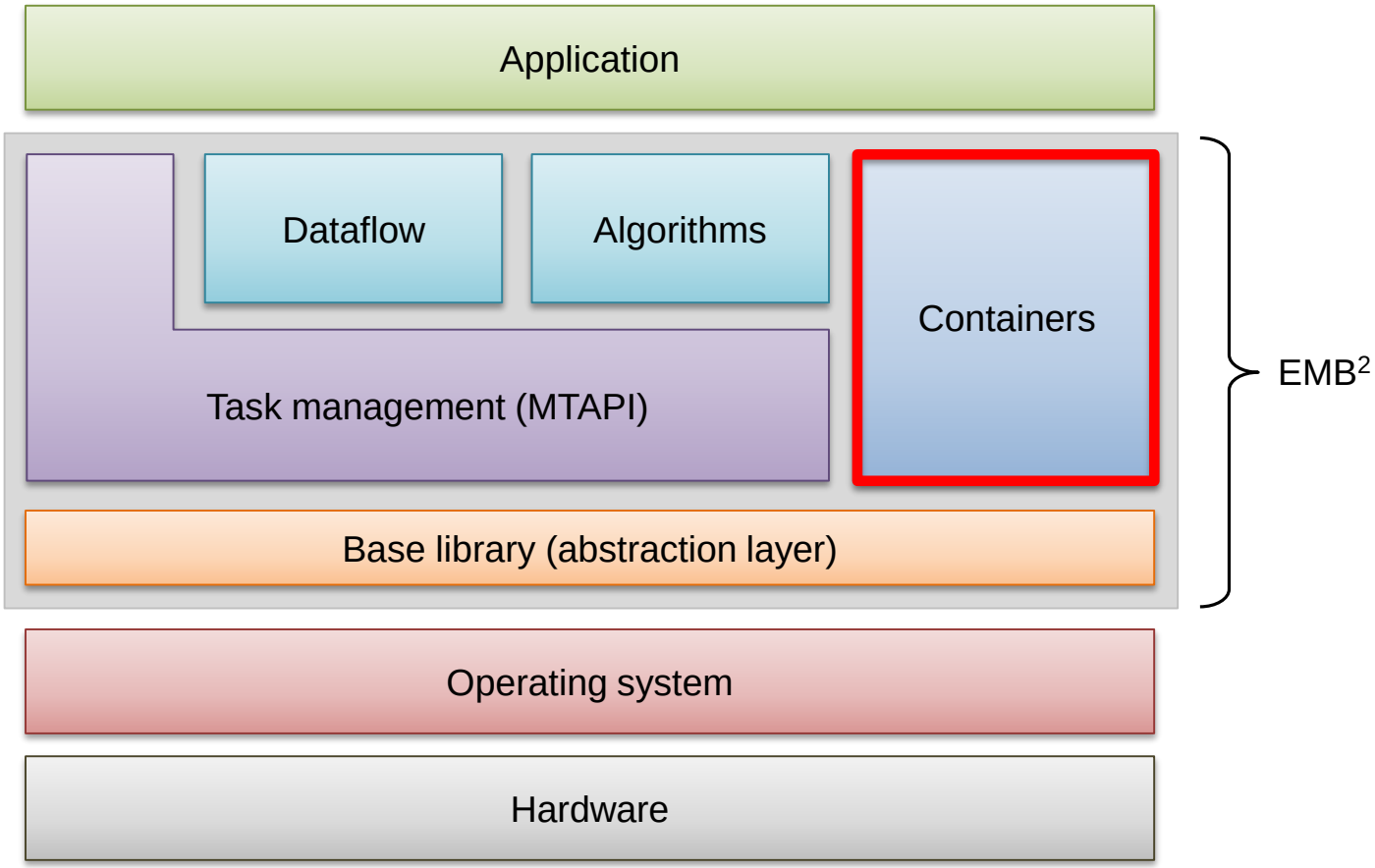
Dataflow Framework

Stream processing

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

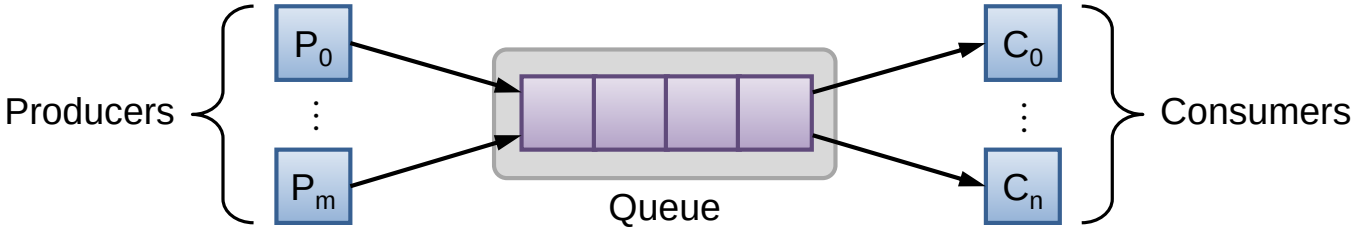


Embedded Multicore Building Blocks
Components



Container Requirements (The Three Commandments)

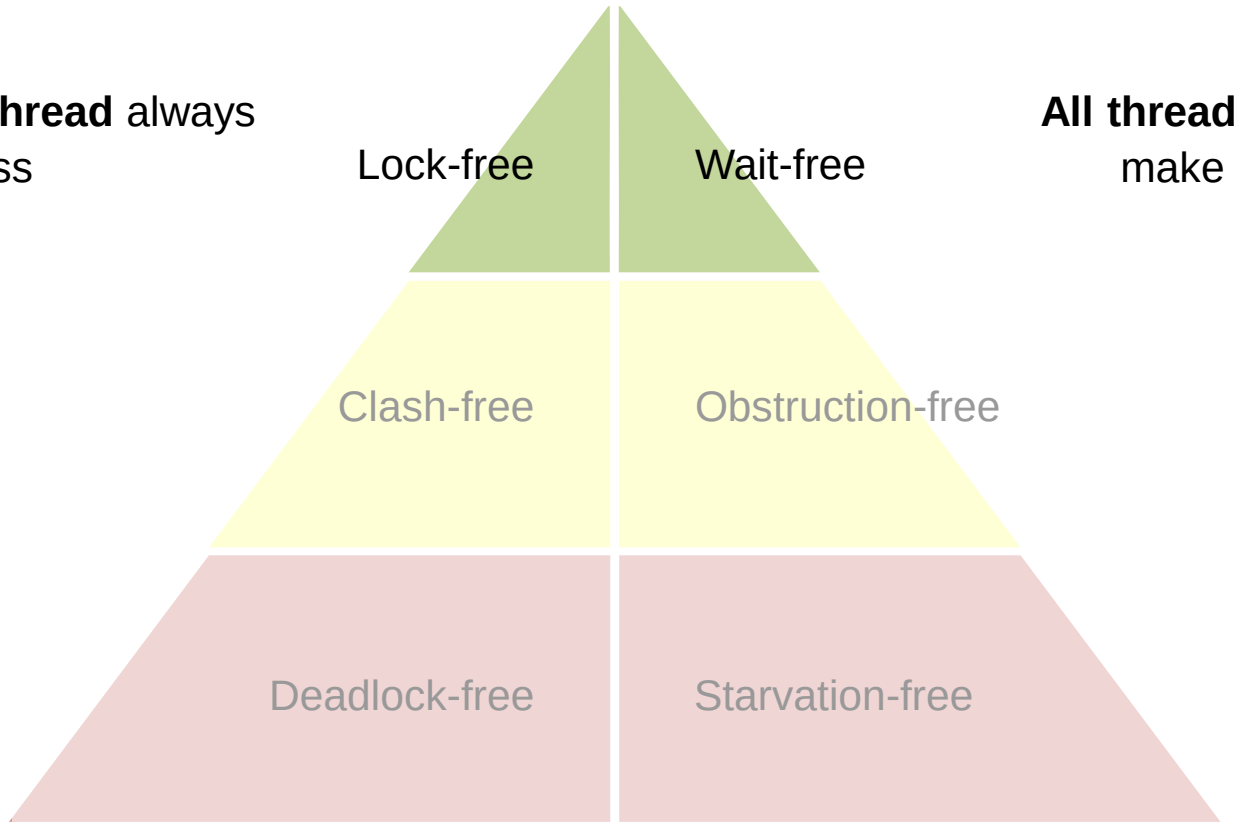
1. **No race conditions** in case of concurrent accesses ⇨ **Thread safety**
2. **No unpredictable delays** in case of contention ⇨ **Progress guarantee**
3. **No dynamic memory allocation** after startup ⇨ **Preallocated memory**



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`	✓	✓	✓

Progress Guarantees

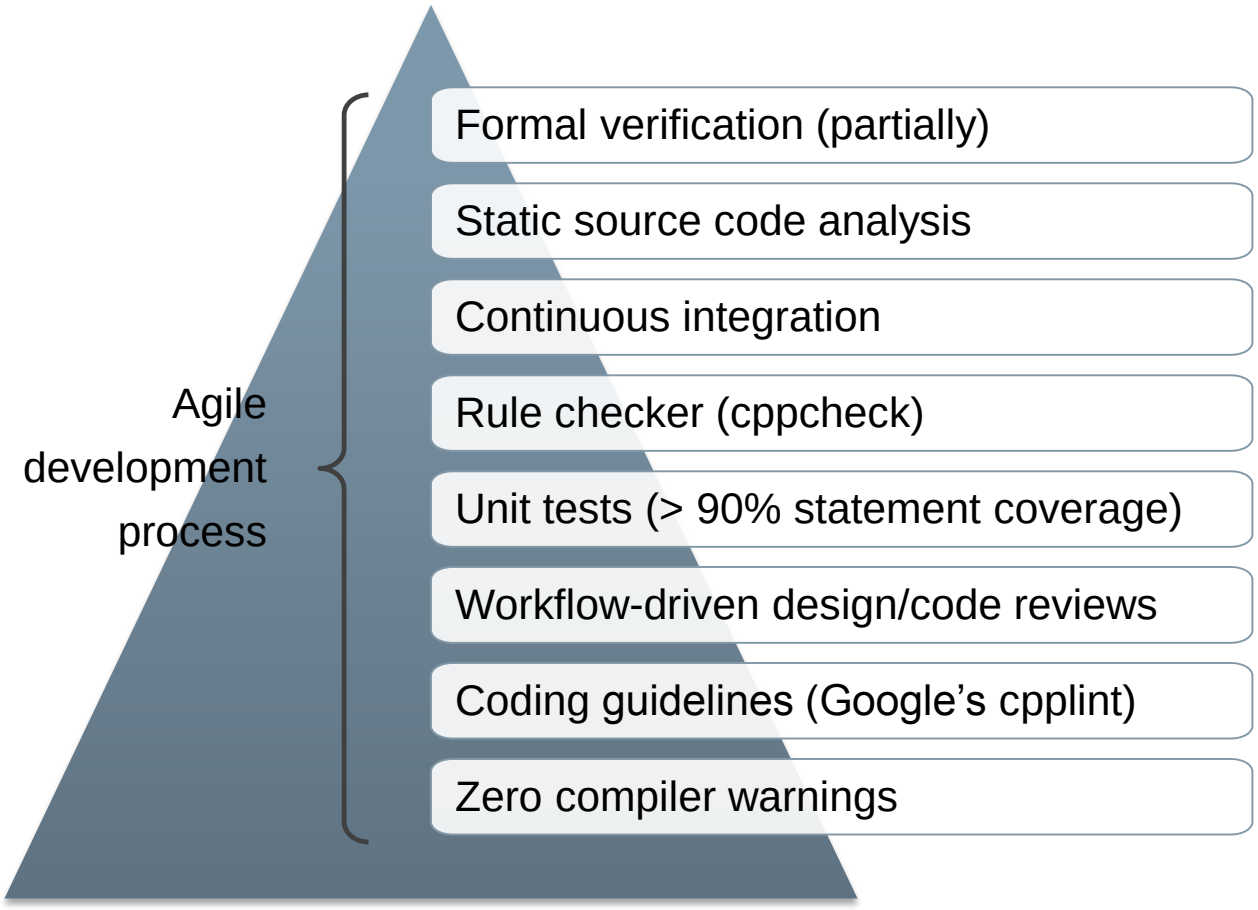
At least one thread always makes progress



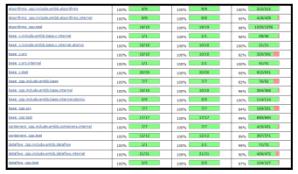
All threads always make progress

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

Code Quality



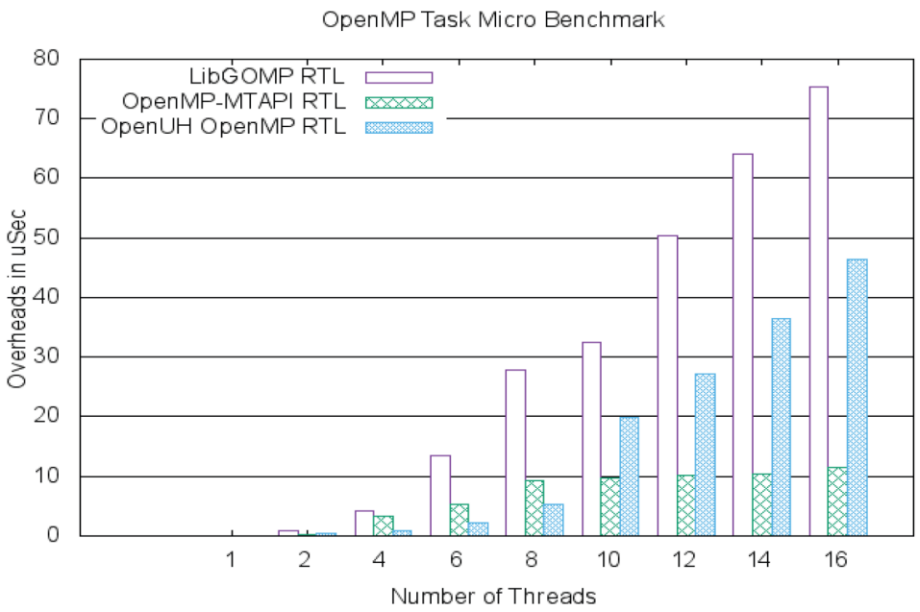
CODESONAR

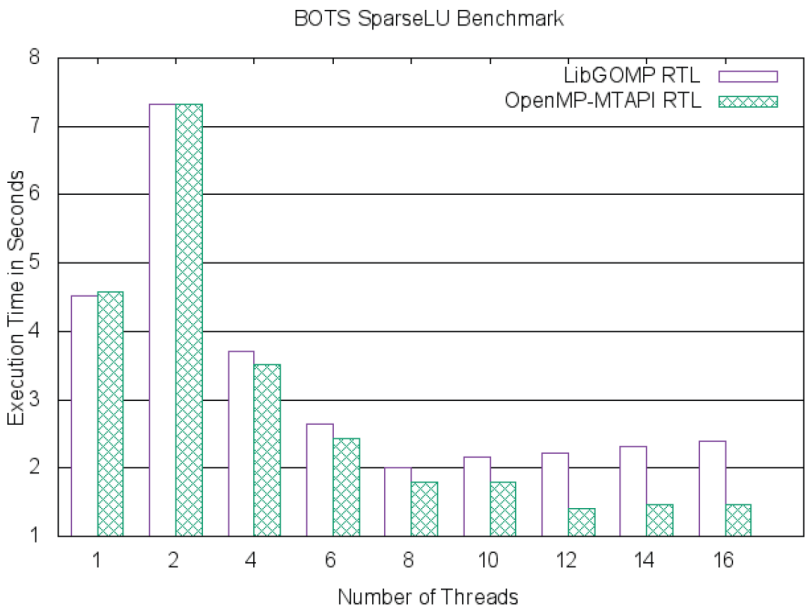


JIRA

Embedded Multicore Building Blocks
Performance Comparison

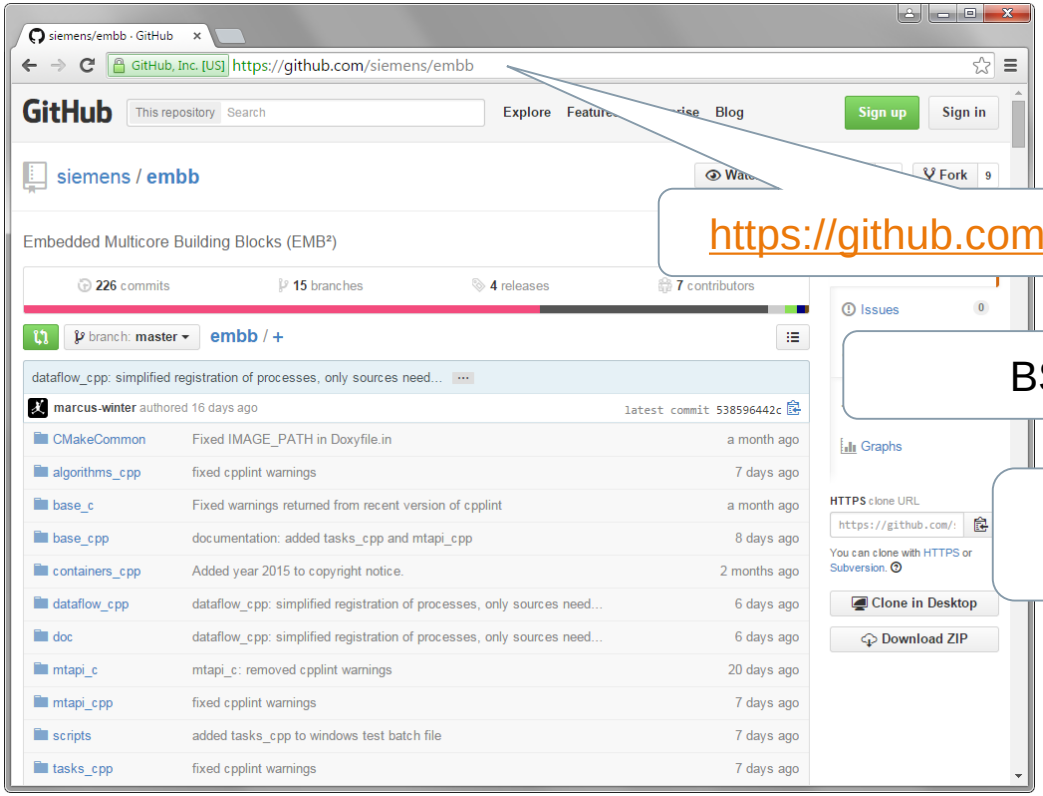
Measurements from the University of Houston show the efficiency of EMB² (green bars):





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, 2015.

Embedded Multicore Building Blocks
Open Source Software



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

BSD 2-clause license

Feedback and contributions are very welcome!

Thank you!