Architectural Specialization for Dynamic Task-Parallel Programs

Christopher Batten

Tao Chen, Shreesha Srinath, G. Edward Suh

Computer Systems Laboratory
School of Electrical and Computer Engineering
Cornell University

ARM Research Summit, Fall 2019

SRC JUMP

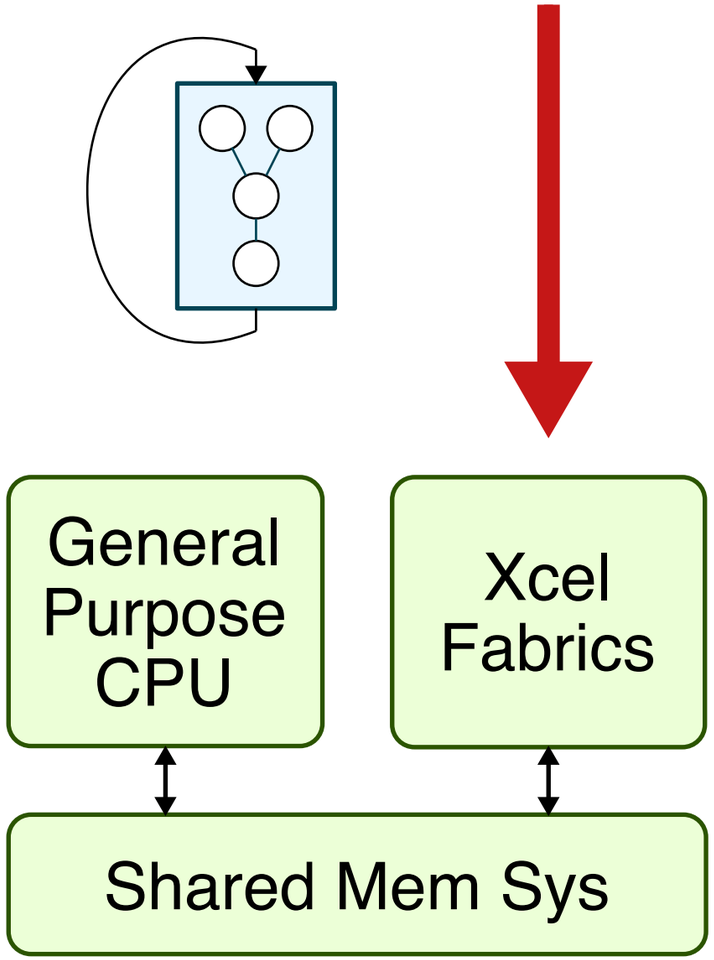
adacenter.org @ADA_Center

This work is supported by the Semiconductor Research Corporation (SRC) and DARPA

ada Applications Driving Architectures

Accelerating *Static* Parallel Algorithms

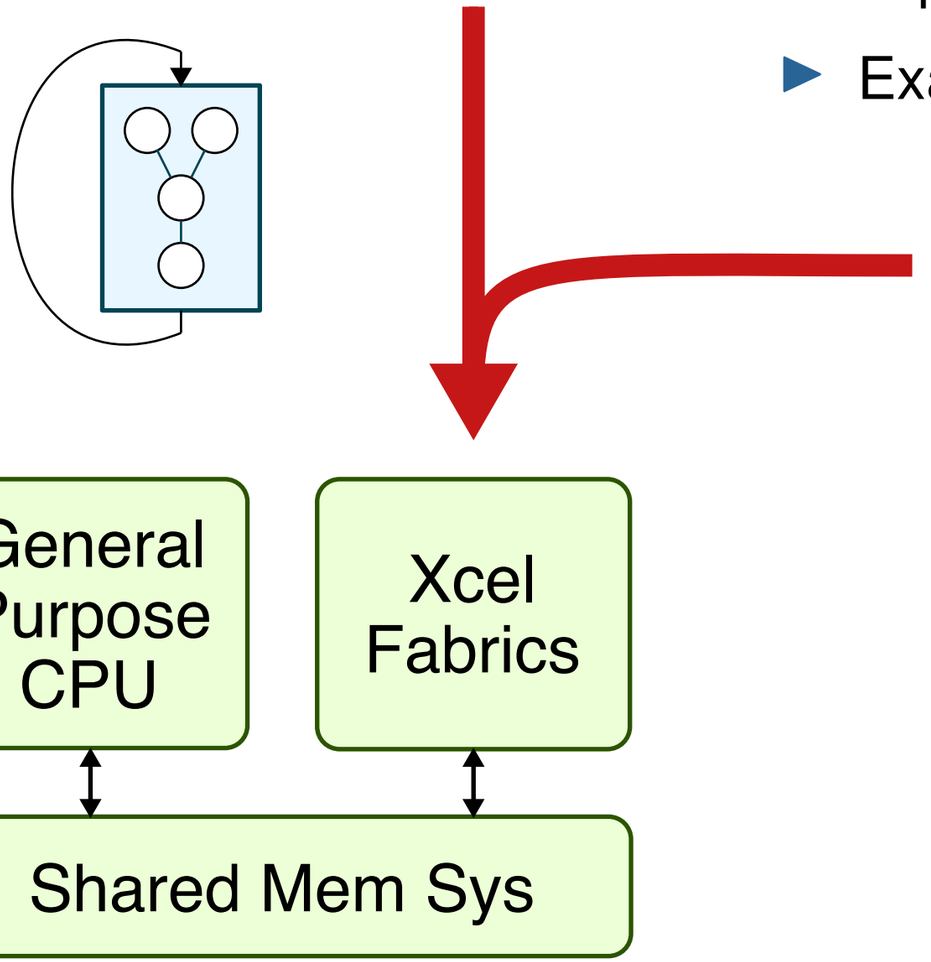
```
for (int i=0; i<n; i++)
  c[i] = a[i] + b[i];
```



- Many programmable accelerators focus on exploiting static loop- or thread-level parallelism
- Examples include FPGAs, CGRAs, Vector, GPGPUs

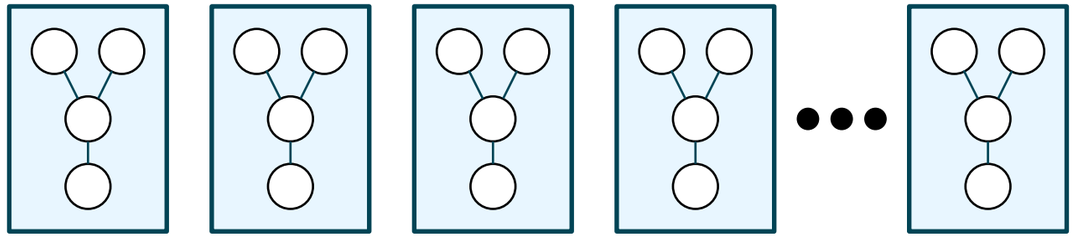
Accelerating *Static* Parallel Algorithms

```
for (int i=0; i<n; i++)
  c[i] = a[i] + b[i];
```



- Many programmable accelerators focus on exploiting static loop- or thread-level parallelism
- Examples include FPGAs, CGRAs, Vector, GPGPUs

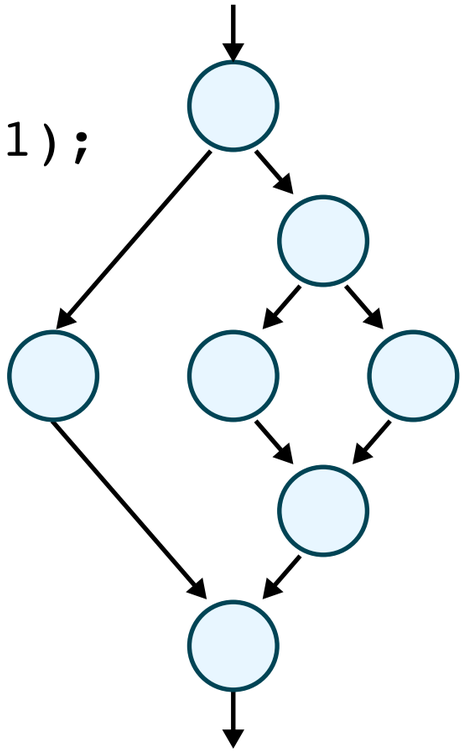
```
__kernel
void vvadd( __global int* c,
            __global int* a,
            __global int* b, int n )
{
  int id = get_global_id(0);
  if ( id < n )
    c[id] = a[id] + b[id];
}
```



Applications Driving Architectures

Programmers Moving from Threads to Tasks

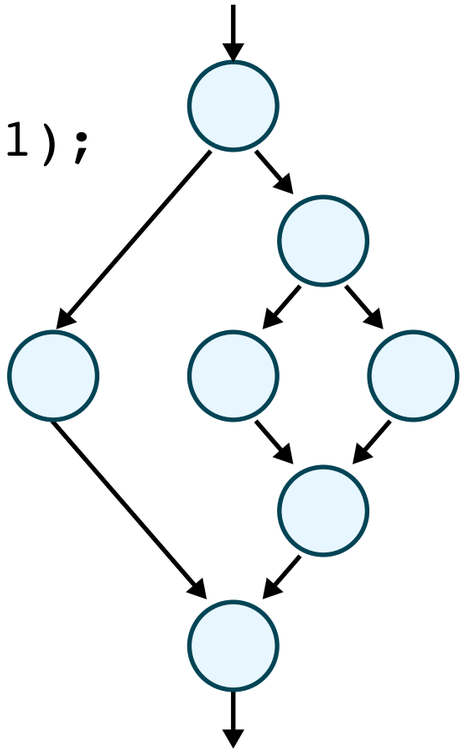
```
int fib( int n )
{
  if (n < 2)
    return n;
  int x = spawn fib(n-1);
  int y = fib(n-2);
  sync;
  return x + y;
}
```



- **Task-parallel programming frameworks** enable creating tasks dynamically as the program executes
  - Intel Cilk Plus, Intel C++ TBB, Microsoft's .NET TPL, Java's Fork/Join, OpenMP

Programmers Moving from Threads to Tasks

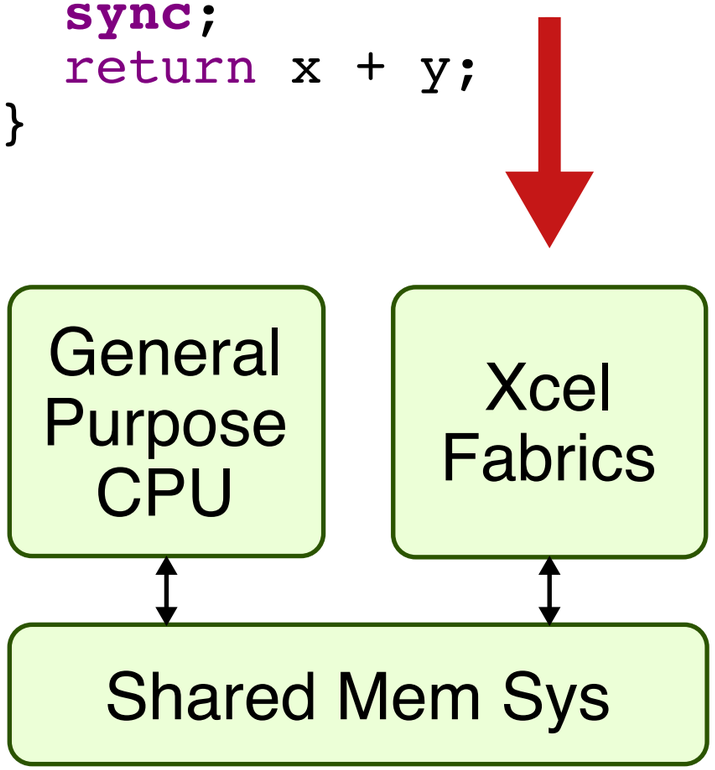
```
int fib( int n )
{
  if (n < 2)
    return n;
  int x = spawn fib(n-1);
  int y = fib(n-2);
  sync;
  return x + y;
}
```

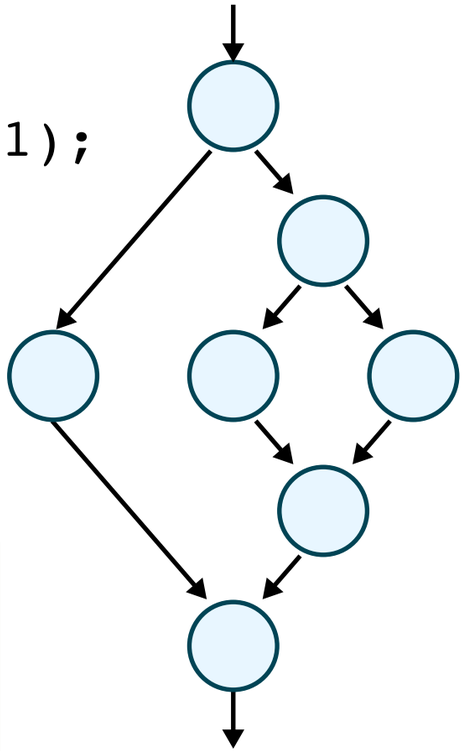


- **Task-parallel programming frameworks** enable creating tasks dynamically as the program executes
  - Intel Cilk Plus, Intel C++ TBB, Microsoft's .NET TPL, Java's Fork/Join, OpenMP
- Benefits of this approach:
  - hierarchical data structures
  - divide-and-conquer algos
  - adaptive algorithms
  - arbitrary nesting, composition
  - efficient in theory and practice
  - automatic load balancing with work stealing

Programmers Moving from Threads to Tasks

```
int fib( int n )
{
  if (n < 2)
    return n;
  int x = spawn fib(n-1);
  int y = fib(n-2);
  sync;
  return x + y;
}
```

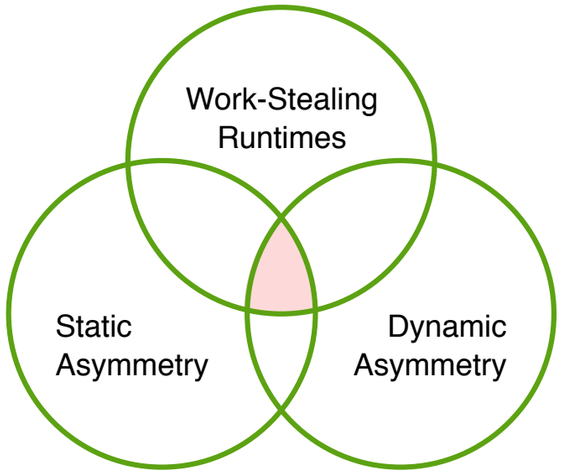


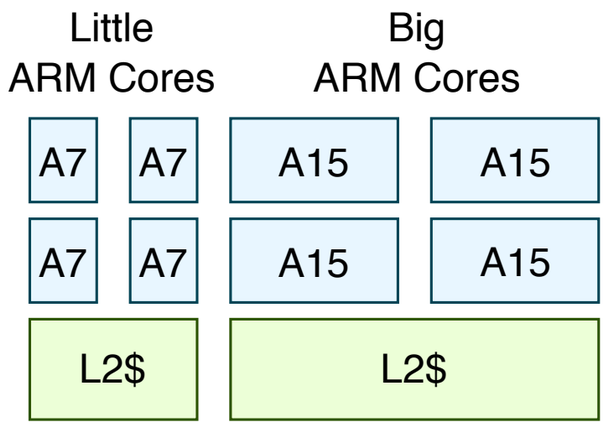


- **Task-parallel programming frameworks** enable creating tasks dynamically as the program executes
  - Intel Cilk Plus, Intel C++ TBB, Microsoft's .NET TPL, Java's Fork/Join, OpenMP
- Benefits of this approach:
  - hierarchical data structures
  - divide-and-conquer algos
  - adaptive algorithms
  - arbitrary nesting, composition
  - efficient in theory and practice
  - automatic load balancing with work stealing

Asymmetry-Aware Work-Stealing

[C. Torng et al., ISCA'16]

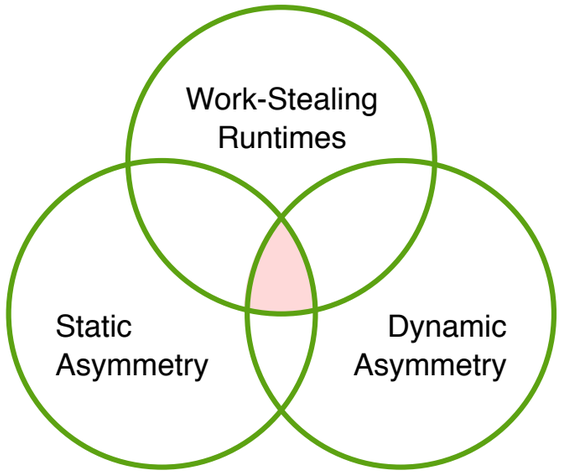


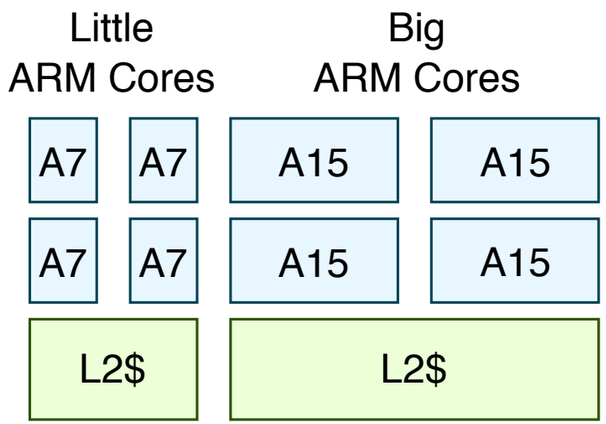


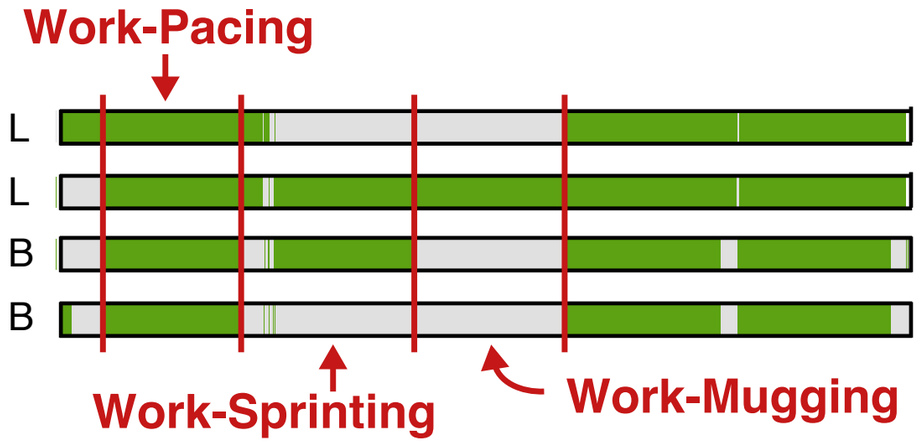
Applications Driving Architectures

Asymmetry-Aware Work-Stealing

[C. Torng et al., ISCA'16]



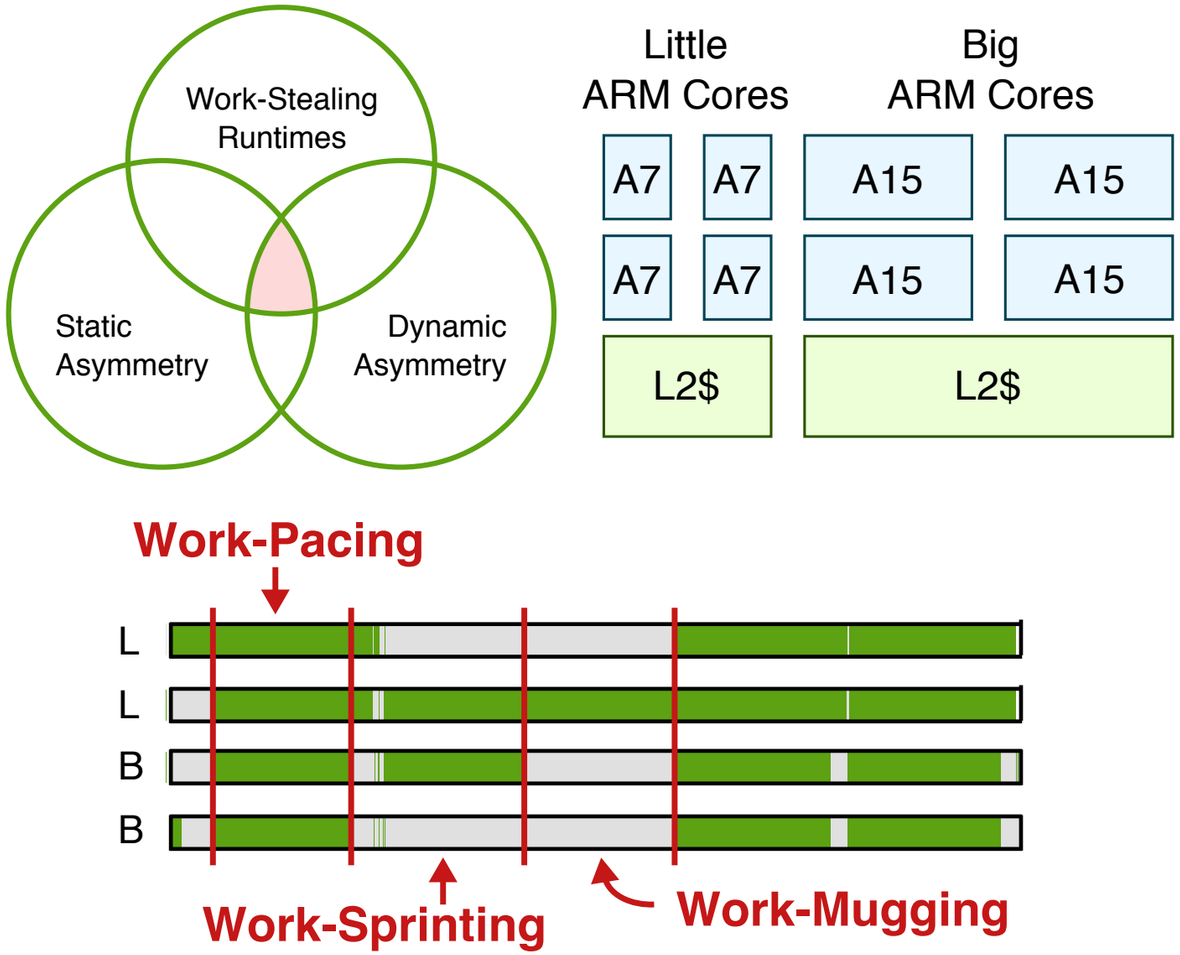




Applications Driving Architectures

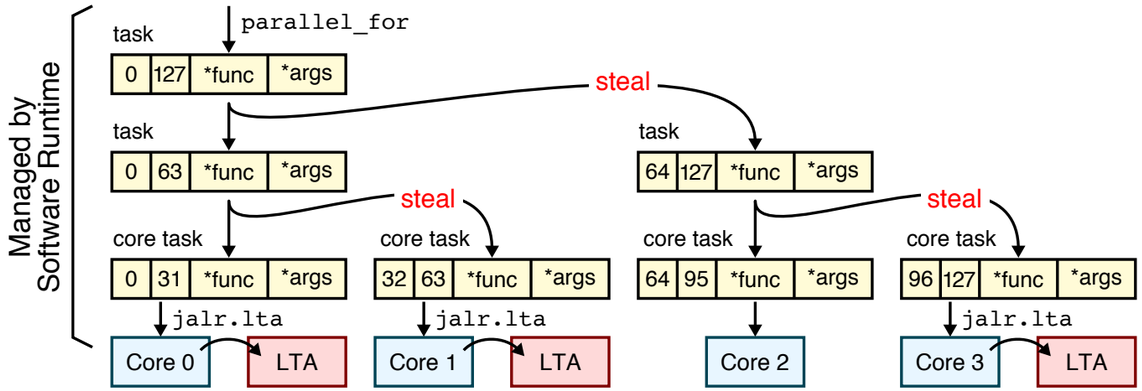
Asymmetry-Aware Work-Stealing

[C. Torng et al., ISCA'16]



Loop-Task Accelerators

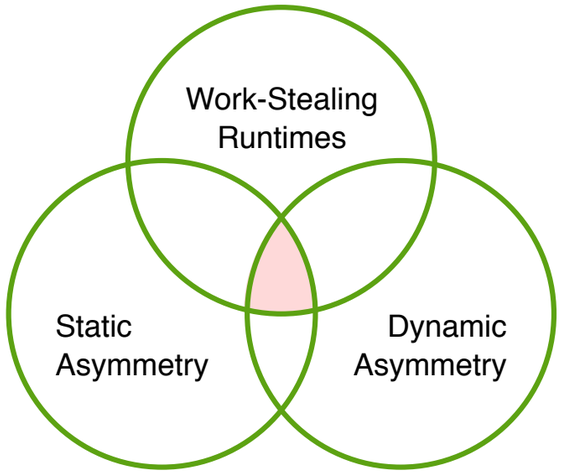
[J. Kim et al., MICRO'17]

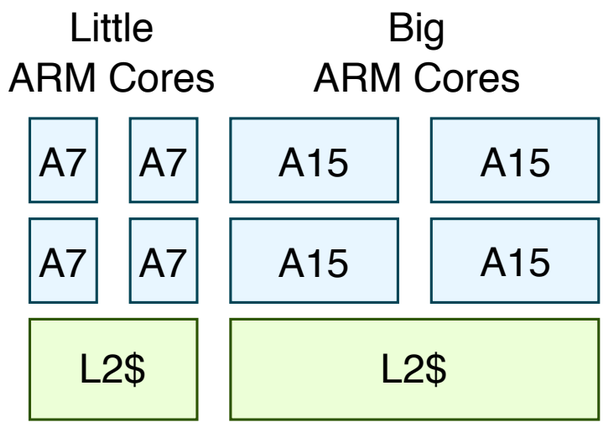


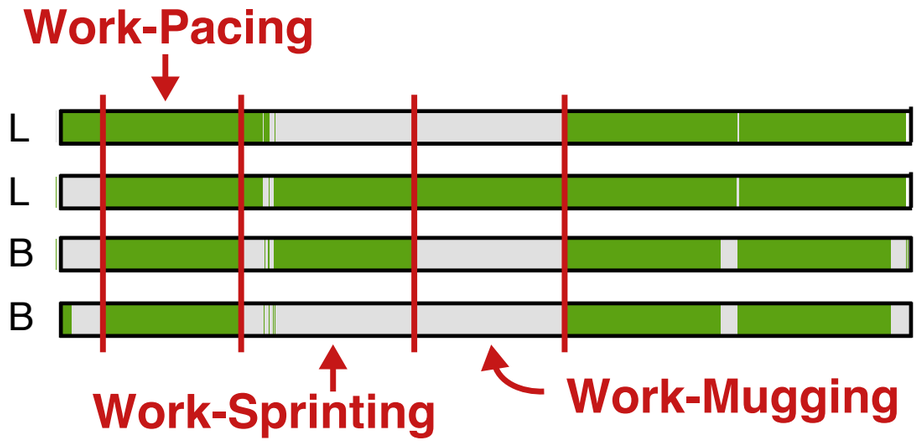
Applications Driving Architectures

Asymmetry-Aware Work-Stealing

[C. Torng et al., ISCA'16]

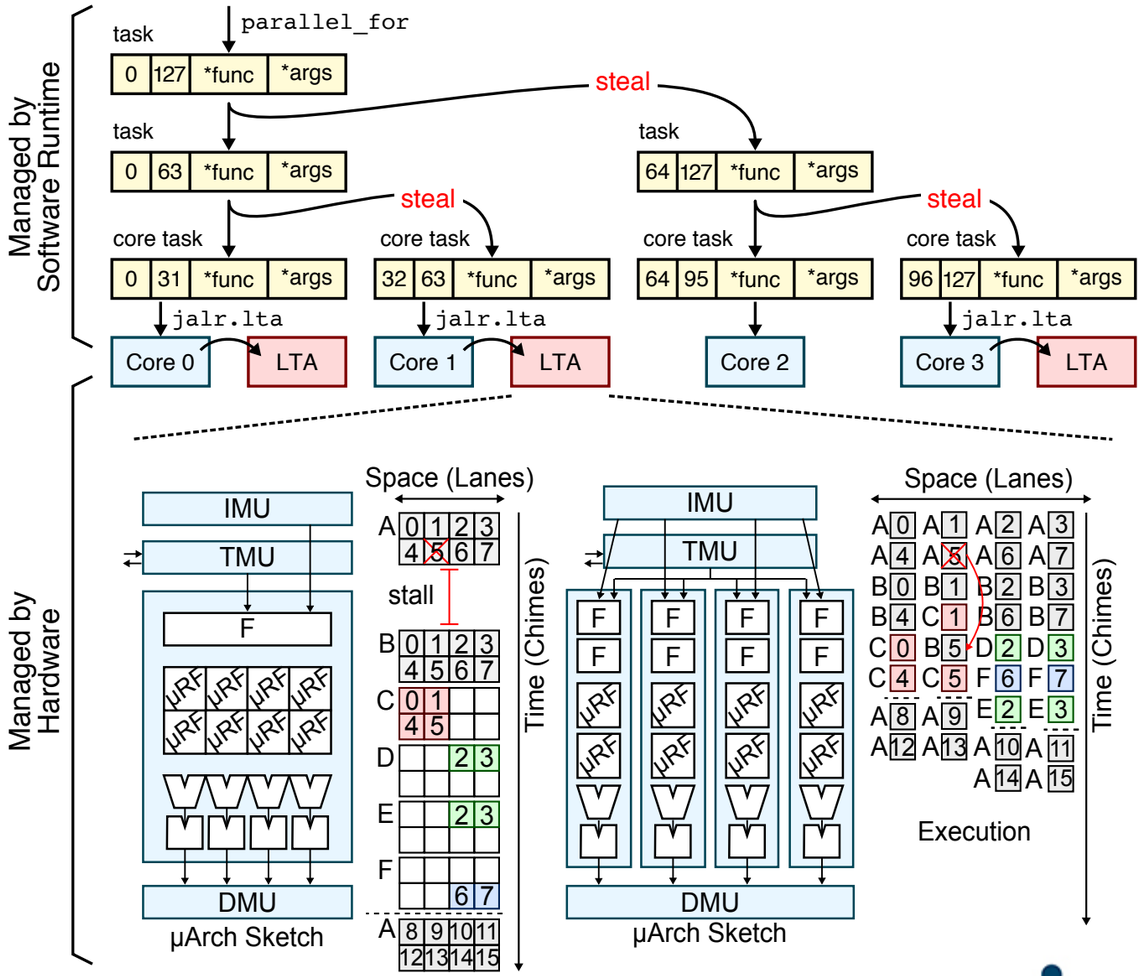






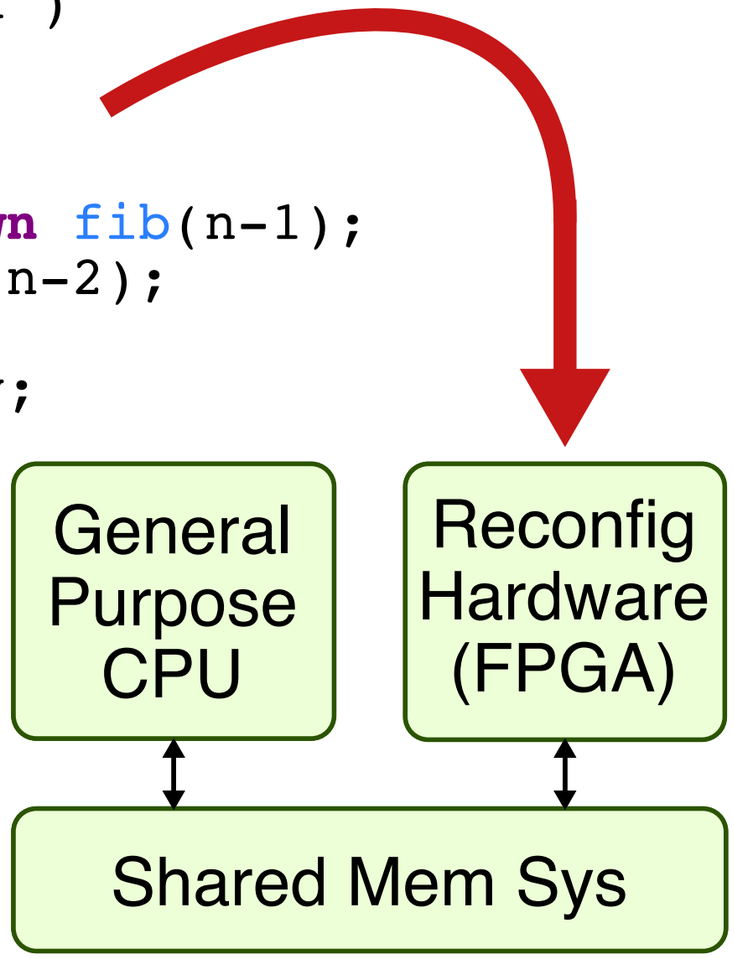
Loop-Task Accelerators

[J. Kim et al., MICRO'17]



BanditXL: Accelerating *Dynamic* Parallel Algorithms on Reconfigurable Hardware

```
int fib( int n )
{
  if (n < 2)
    return n;
  int x = spawn fib(n-1);
  int y = fib(n-2);
  sync;
  return x + y;
}
```



Tao Chen
Shreesha Srinath
Christopher Batten
G. Edward Suh
MICRO'18

Motivation

Computation Model

Accelerator Architecture

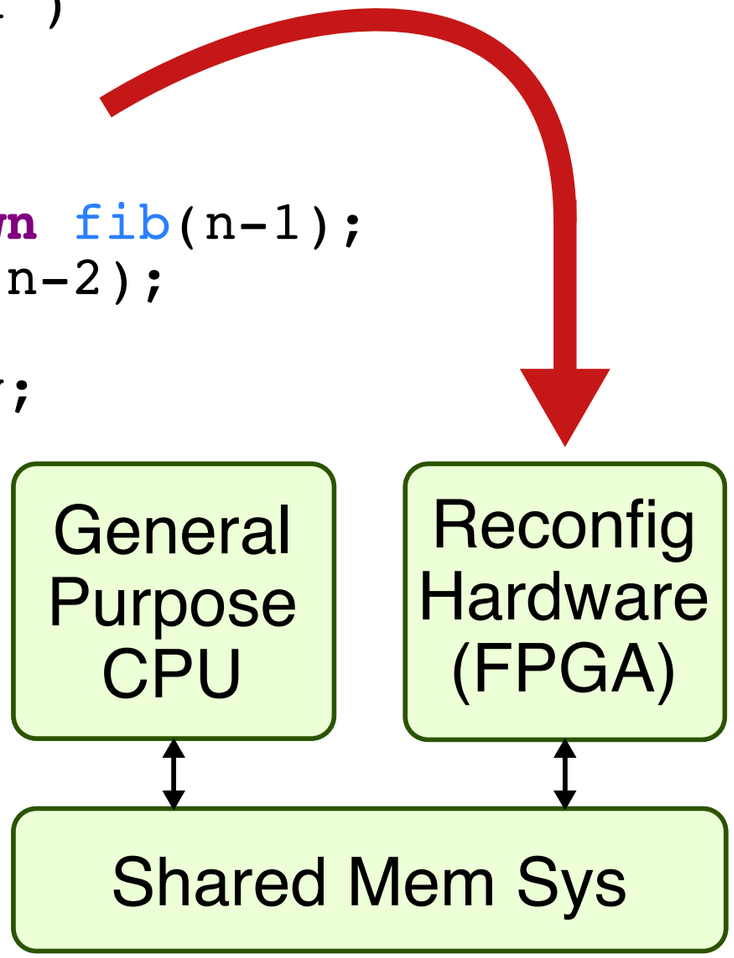
Design Methodology

Evaluation

Ongoing Work

BanditXL: Accelerating *Dynamic* Parallel Algorithms on Reconfigurable Hardware

```
int fib( int n )
{
  if (n < 2)
    return n;
  int x = spawn fib(n-1);
  int y = fib(n-2);
  sync;
  return x + y;
}
```



Tao Chen
Shreesha Srinath
Christopher Batten
G. Edward Suh
MICRO'18

Motivation

Computation Model

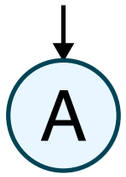
Accelerator Architecture

Design Methodology

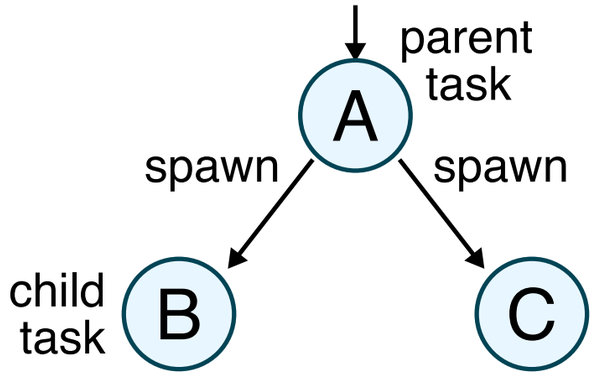
Evaluation

Ongoing Work

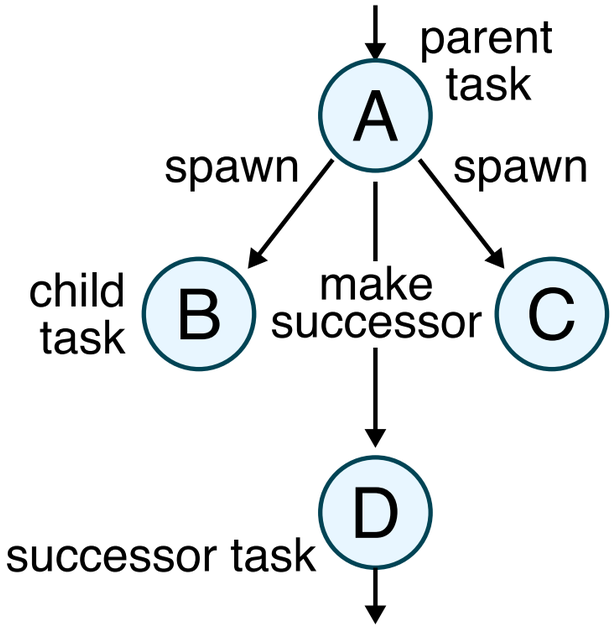
Explicit Continuation Passing



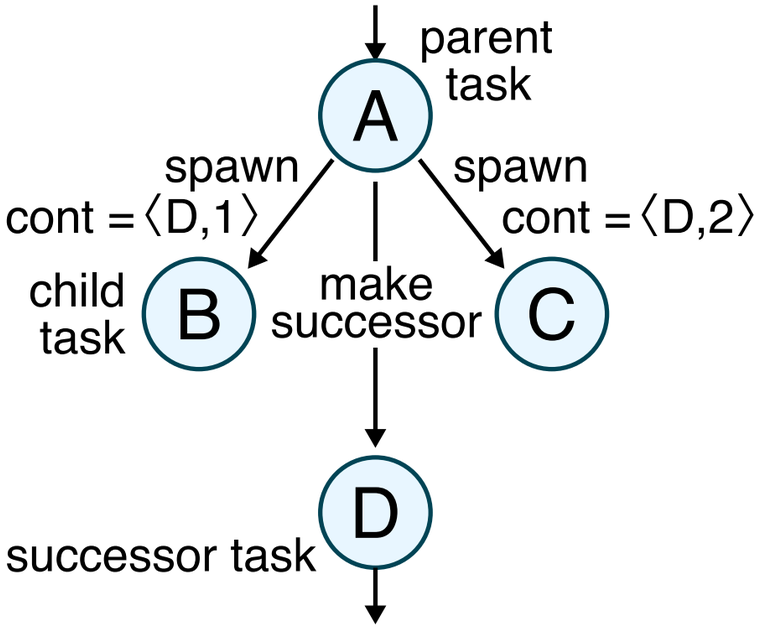
Explicit Continuation Passing



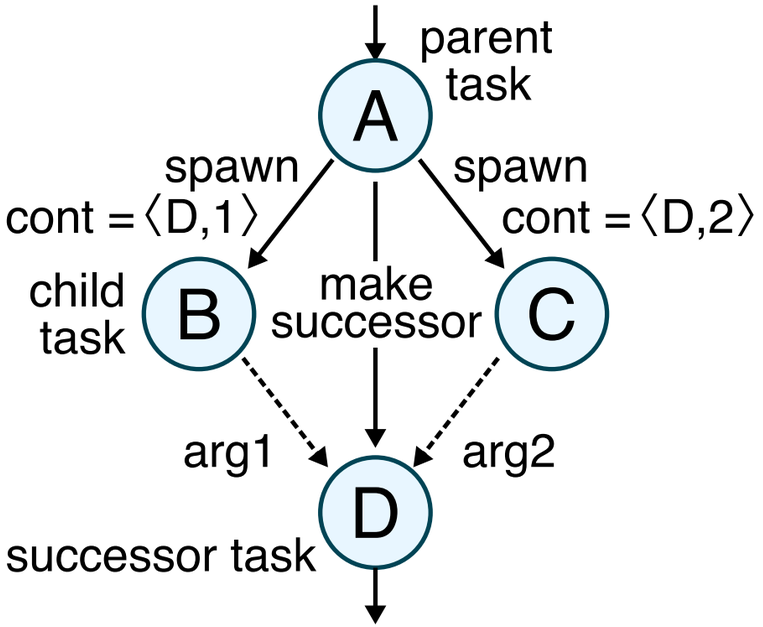
Explicit Continuation Passing



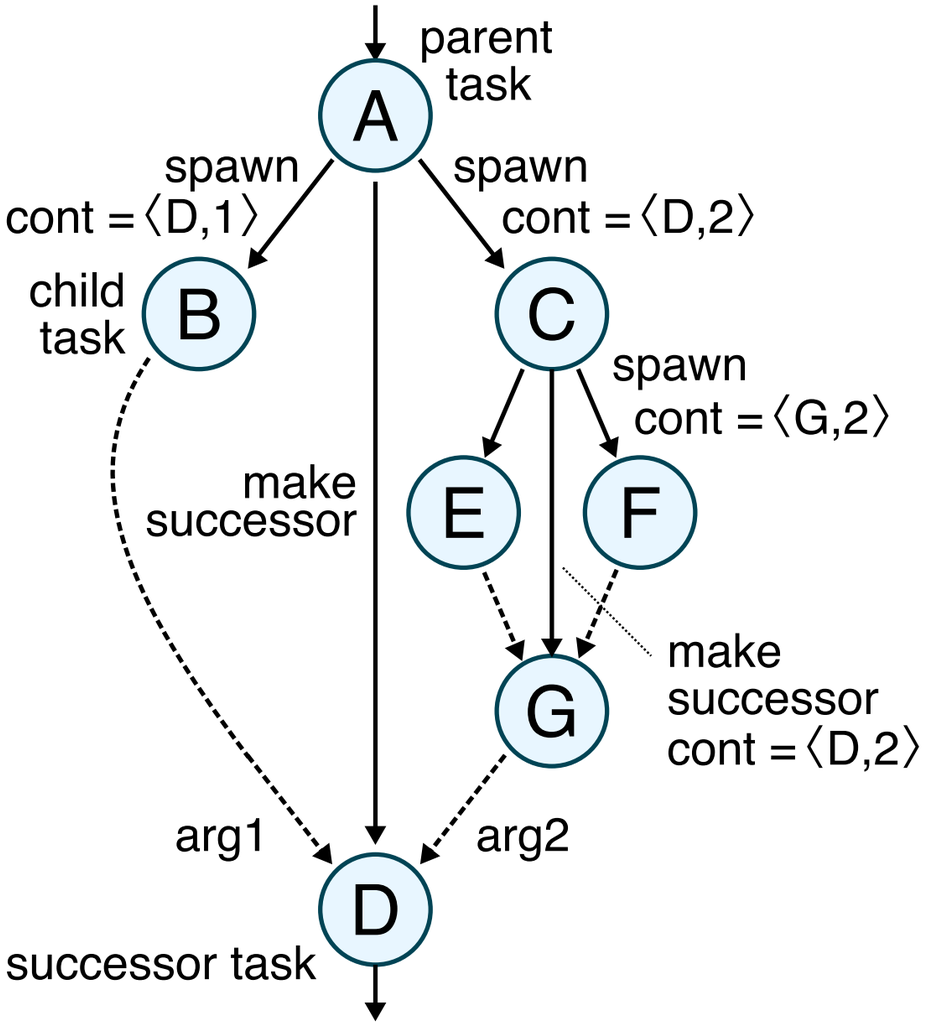
Explicit Continuation Passing



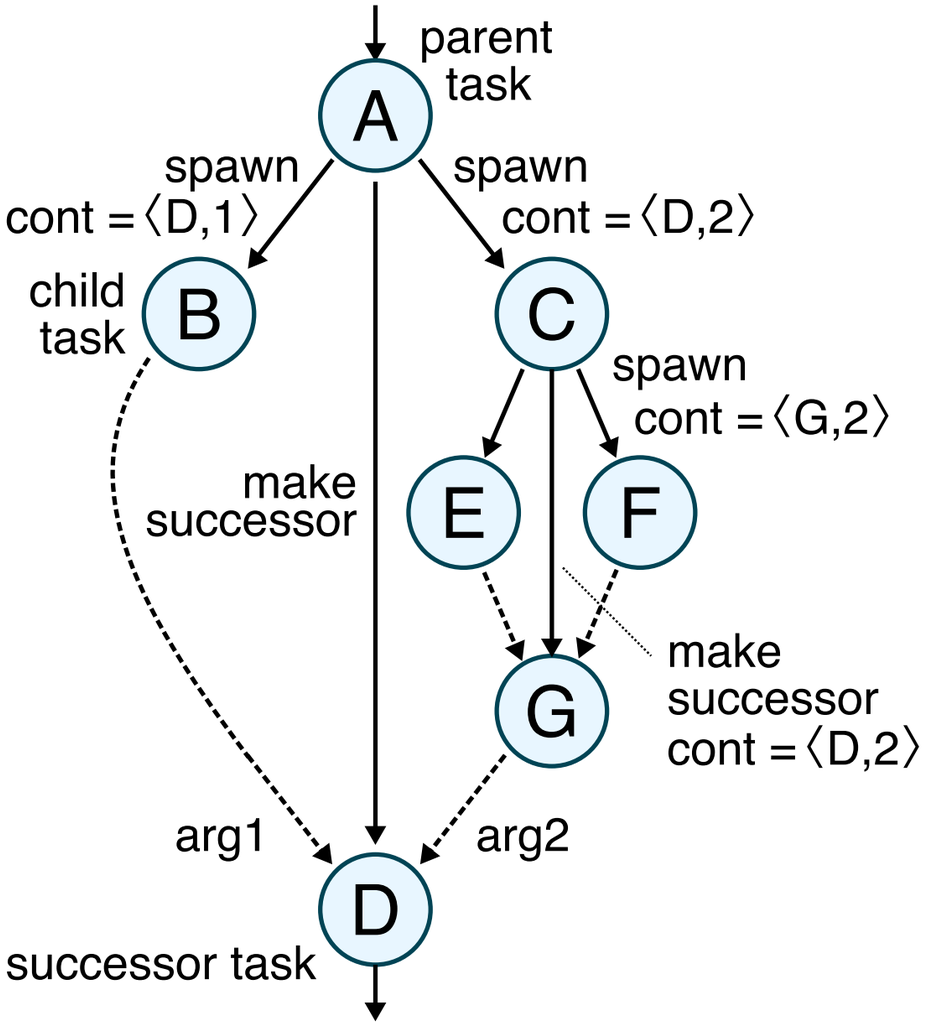
Explicit Continuation Passing



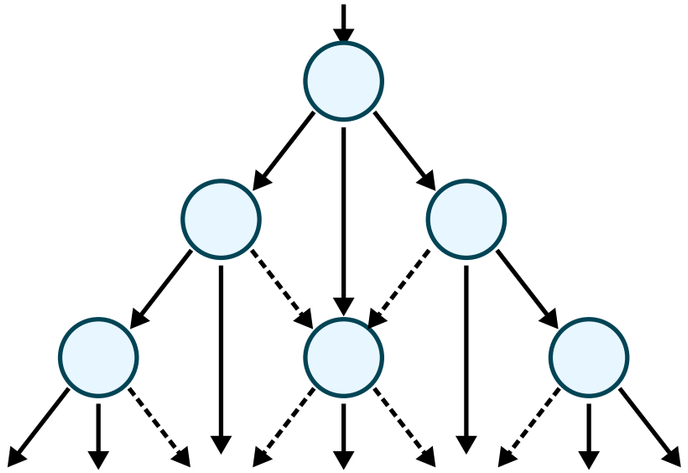
Explicit Continuation Passing



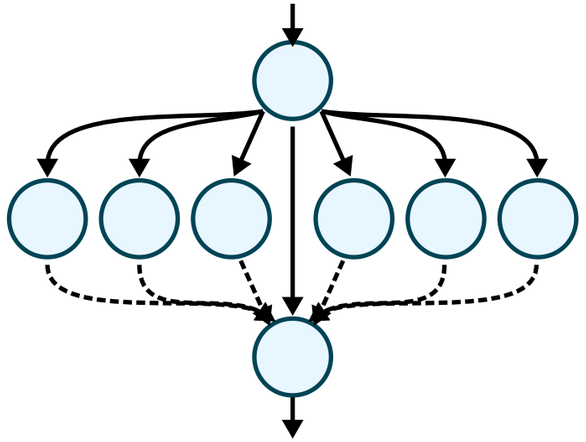
Explicit Continuation Passing



Fork/Join Pattern

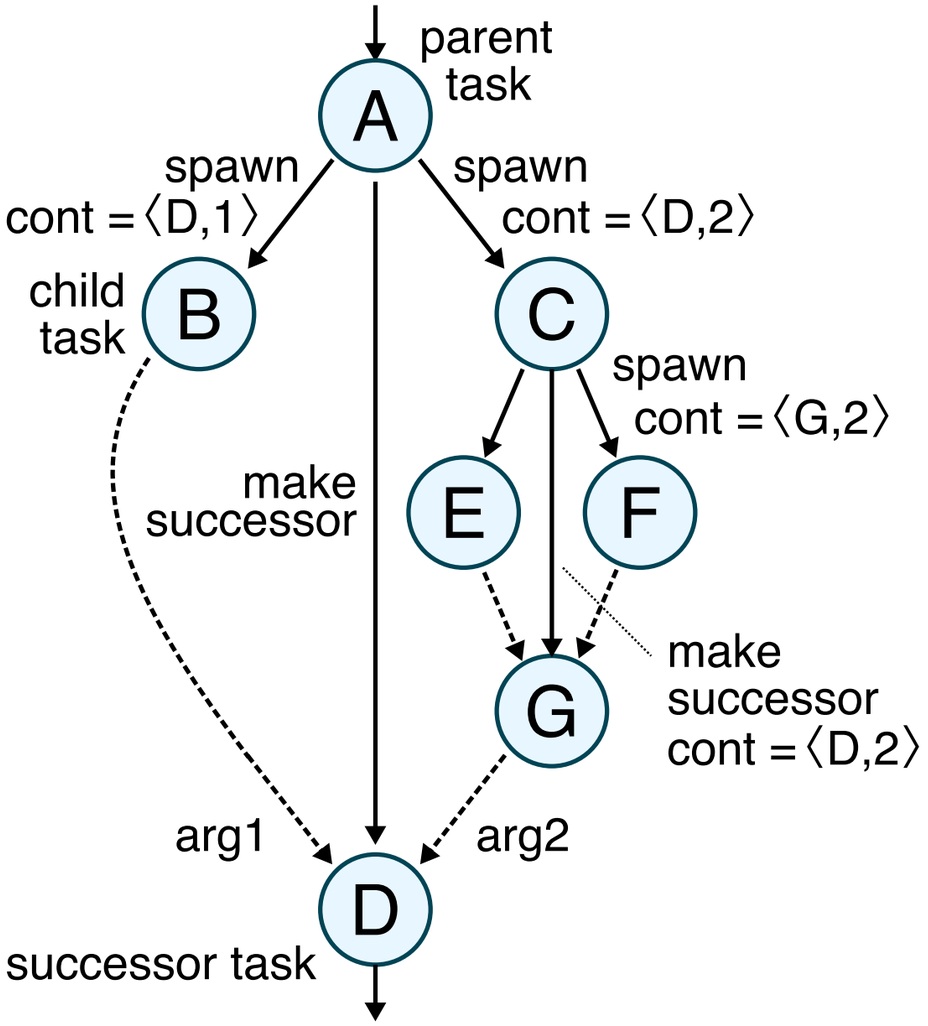


Data-Flow Pattern

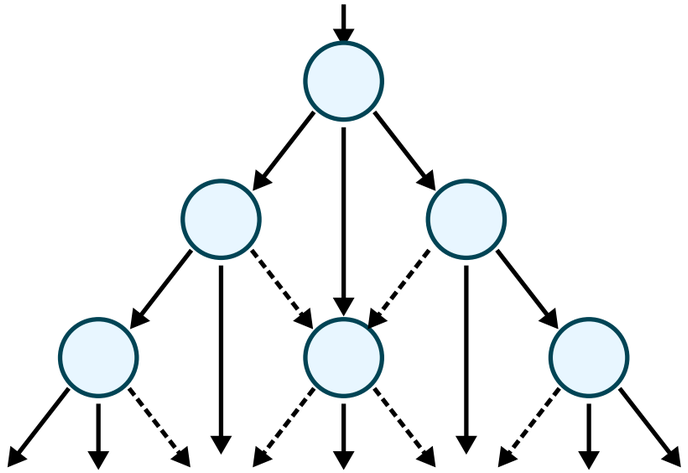


Data-Parallel Pattern

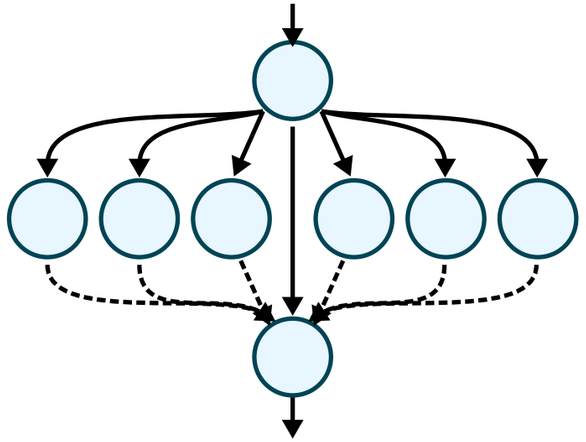
Explicit Continuation Passing



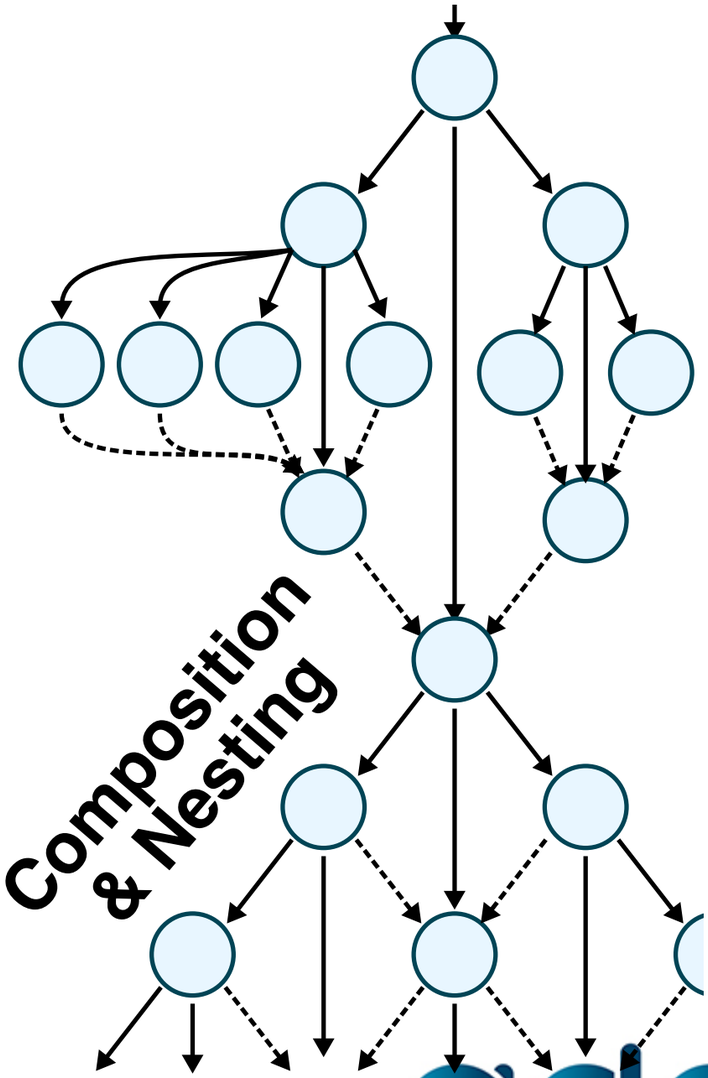
Fork/Join Pattern



Data-Flow Pattern



Data-Parallel Pattern



Composition & Nesting

Explicit Continuation Passing and More ...

```
task fib( cont int k, int n ) {
  if ( n < 2 )
    send_argument( k, n );
  else {
    cont int x, y;
    spawn_next sum( k, ?x, ?y );
    spawn      fib( x, n-1 );
    spawn      fib( y, n-2 );
  }
}

task sum( cont int k, int x, int y ) {
  send_argument( k, x+y );
}
```

- Cilk-1 used explicit continuation passing (JPDC'96)

Explicit Continuation Passing and More ...

```
task fib( cont int k, int n ) {
  if ( n < 2 )
    send_argument( k, n );
  else {
    cont int x, y;
    spawn_next sum( k, ?x, ?y );
    spawn      fib( x, n-1 );
    spawn      fib( y, n-2 );
  }
}

task sum( cont int k, int x, int y ) {
  send_argument( k, x+y );
}
```

```
int fib( int n ) {
  if (n < 2) return n;
  int x = spawn fib(n-1);
  int y = fib(n-2);
  sync;
  return x + y;
}
```

- Cilk-1 used explicit continuation passing (JPDC'96)
- Cilk-5 used call/return semantics for parallelism (PLDI'98)

Explicit Continuation Passing and More ...

```
task fib( cont int k, int n ) {
  if ( n < 2 )
    send_argument( k, n );
  else {
    cont int x, y;
    spawn_next sum( k, ?x, ?y );
    spawn      fib( x, n-1 );
    spawn      fib( y, n-2 );
  }
}

task sum( cont int k, int x, int y ) {
  send_argument( k, x+y );
}
```

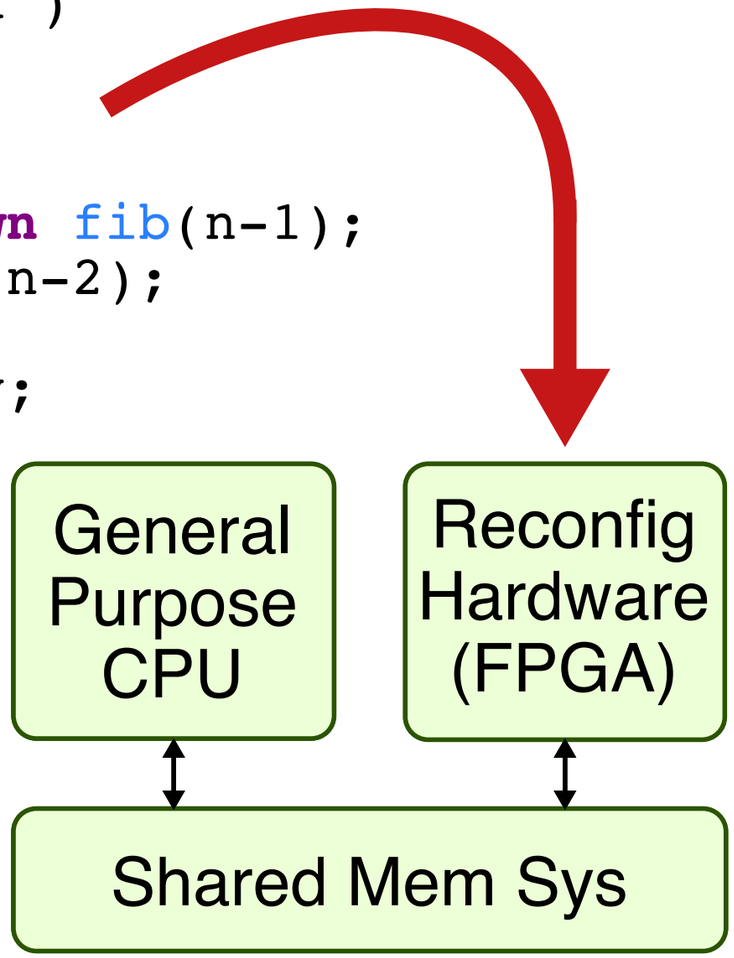
```
int fib( int n ) {
  if (n < 2) return n;
  int x = spawn fib(n-1);
  int y = fib(n-2);
  sync;
  return x + y;
}
```

```
int fib( int n ) {
  if (n < 2) return n;
  int x, y;
  tbb::parallel_invoke(
    [&]{ x = fib(n-1); }
    [&]{ y = fib(n-2); }
  });
  return x + y;
```

- Cilk-1 used explicit continuation passing (JPDC'96)
- Cilk-5 used call/return semantics for parallelism (PLDI'98)
- Many library-based frameworks also exist

BanditXL: Accelerating *Dynamic* Parallel Algorithms on Reconfigurable Hardware

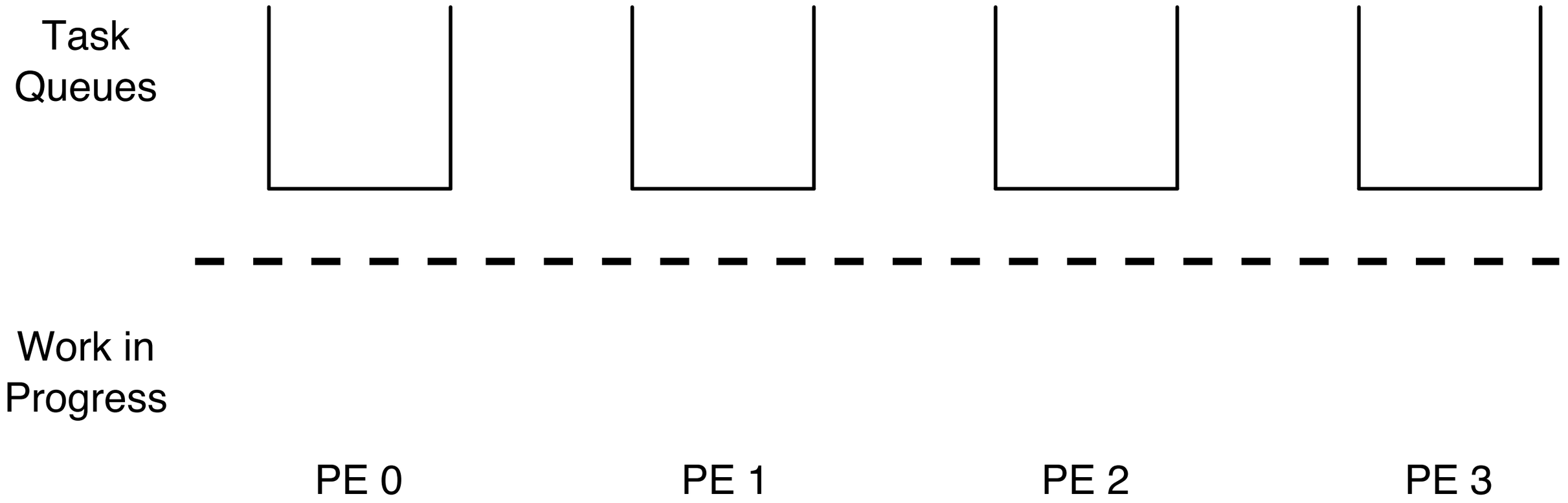
```
int fib( int n )
{
  if (n < 2)
    return n;
  int x = spawn fib(n-1);
  int y = fib(n-2);
  sync;
  return x + y;
}
```



Tao Chen
Shreesha Srinath
Christopher Batten
G. Edward Suh
MICRO'18

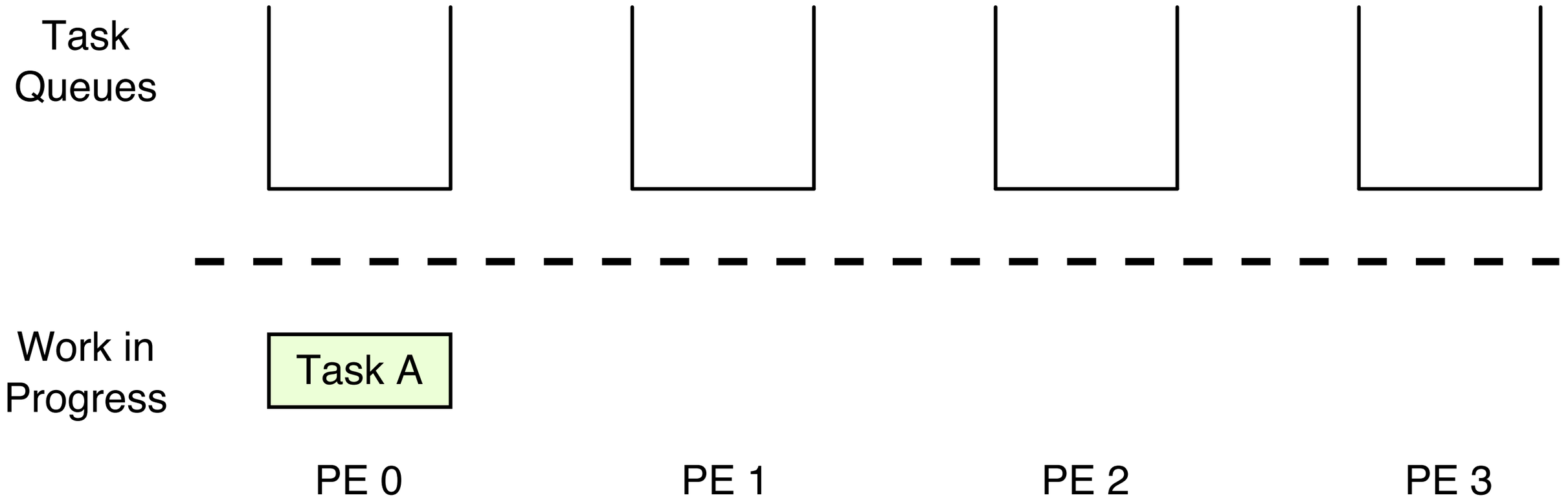
Motivation
Computation Model
Accelerator Architecture
Design Methodology
Evaluation
Ongoing Work

Scheduling Tasks with Work Stealing



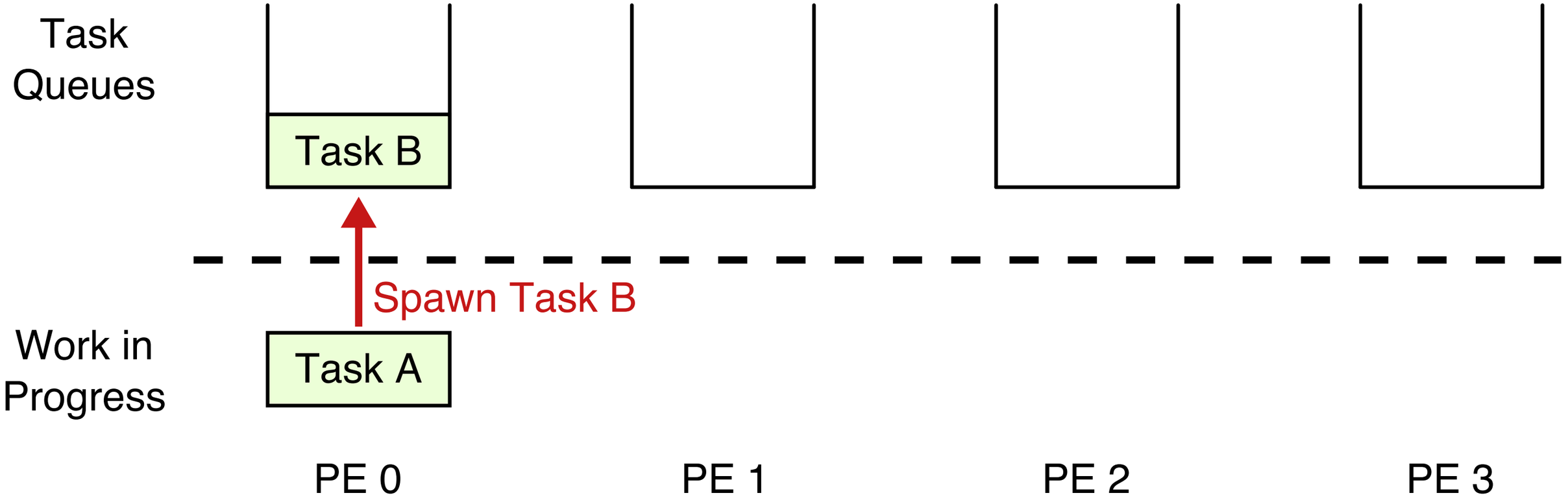
- Work stealing has good performance, space requirements, and communication overheads in both theory and practice

Scheduling Tasks with Work Stealing



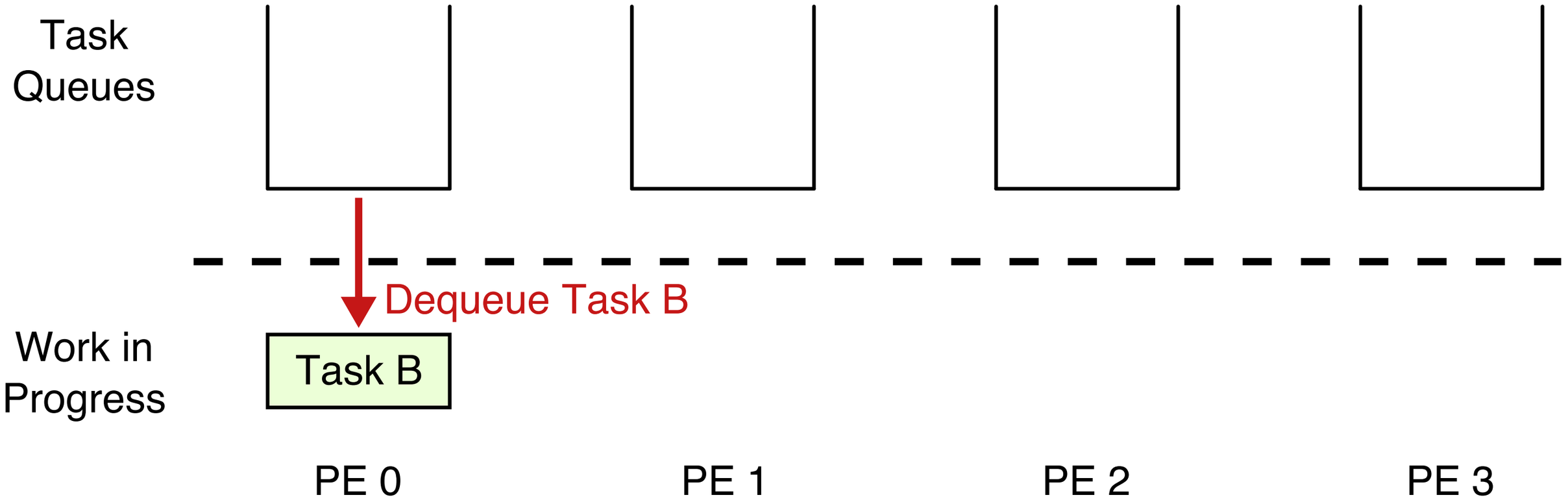
- Work stealing has good performance, space requirements, and communication overheads in both theory and practice

Scheduling Tasks with Work Stealing



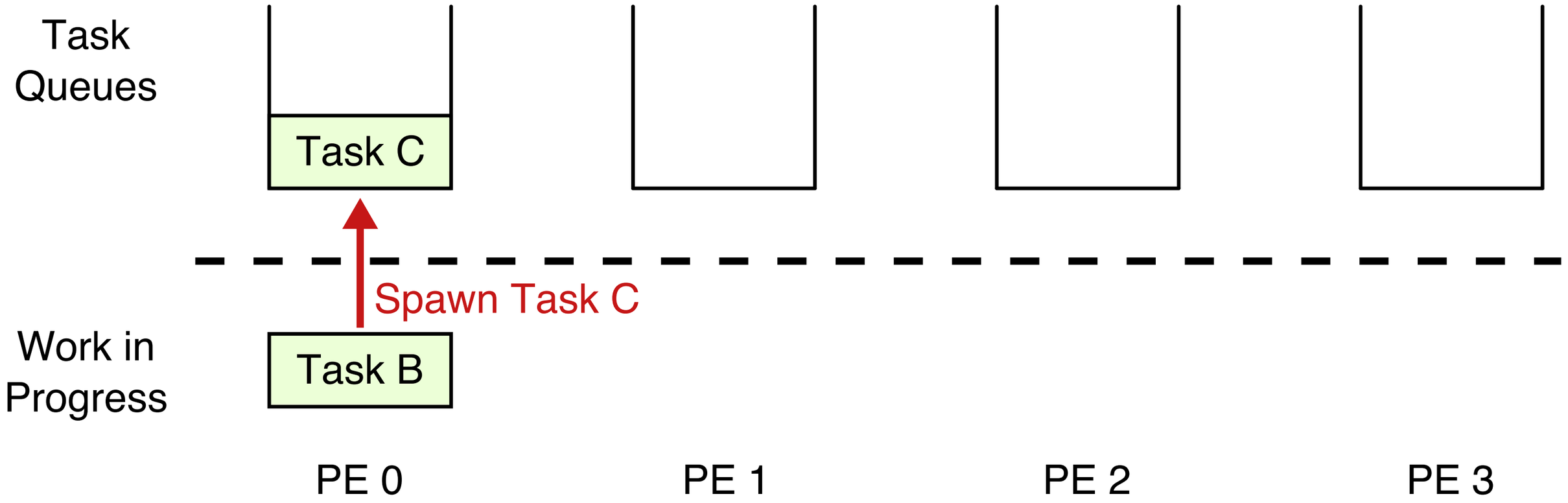
- Work stealing has good performance, space requirements, and communication overheads in both theory and practice

Scheduling Tasks with Work Stealing



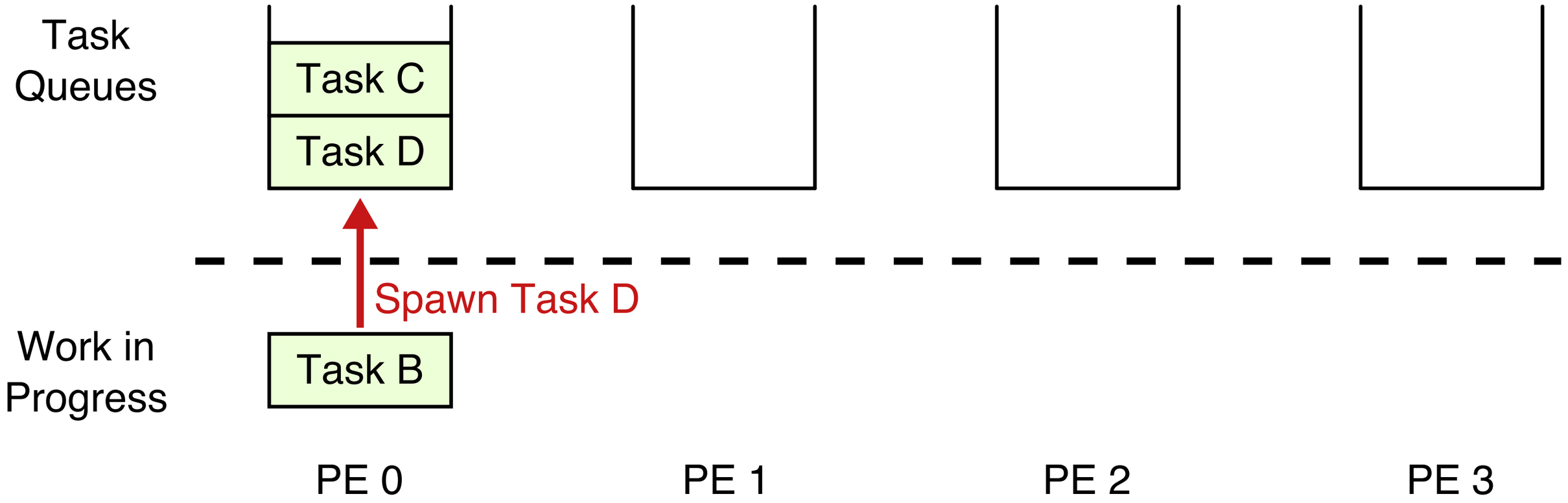
- Work stealing has good performance, space requirements, and communication overheads in both theory and practice

Scheduling Tasks with Work Stealing



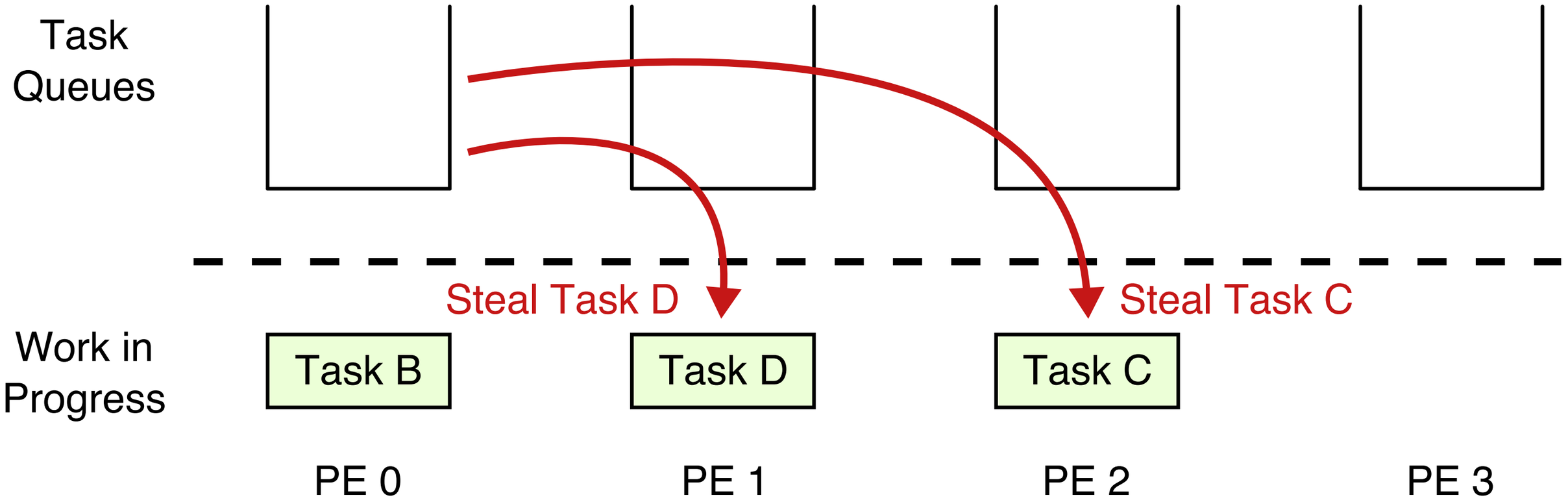
- Work stealing has good performance, space requirements, and communication overheads in both theory and practice

Scheduling Tasks with Work Stealing



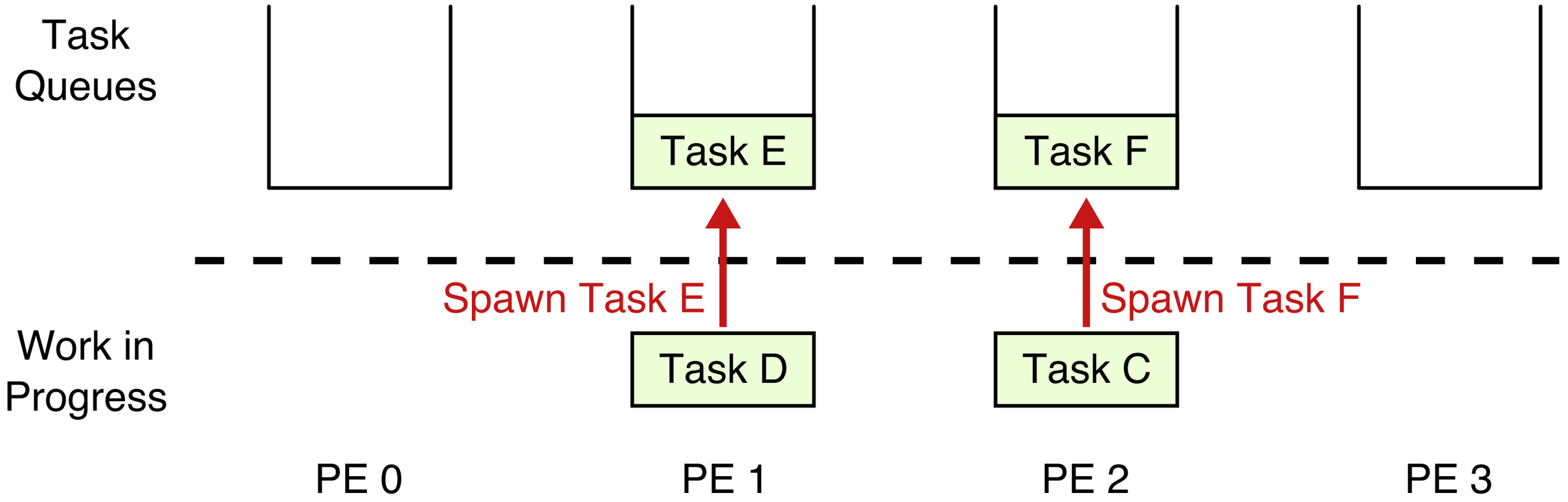
- Work stealing has good performance, space requirements, and communication overheads in both theory and practice

Scheduling Tasks with Work Stealing



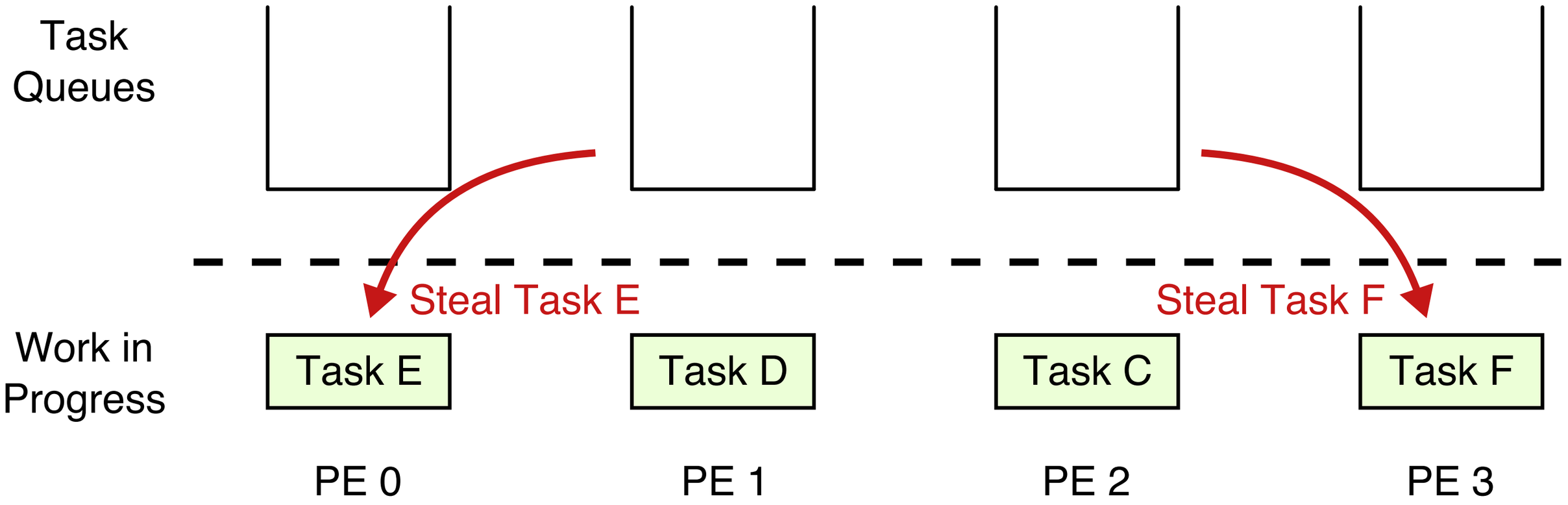
- Work stealing has good performance, space requirements, and communication overheads in both theory and practice

Scheduling Tasks with Work Stealing



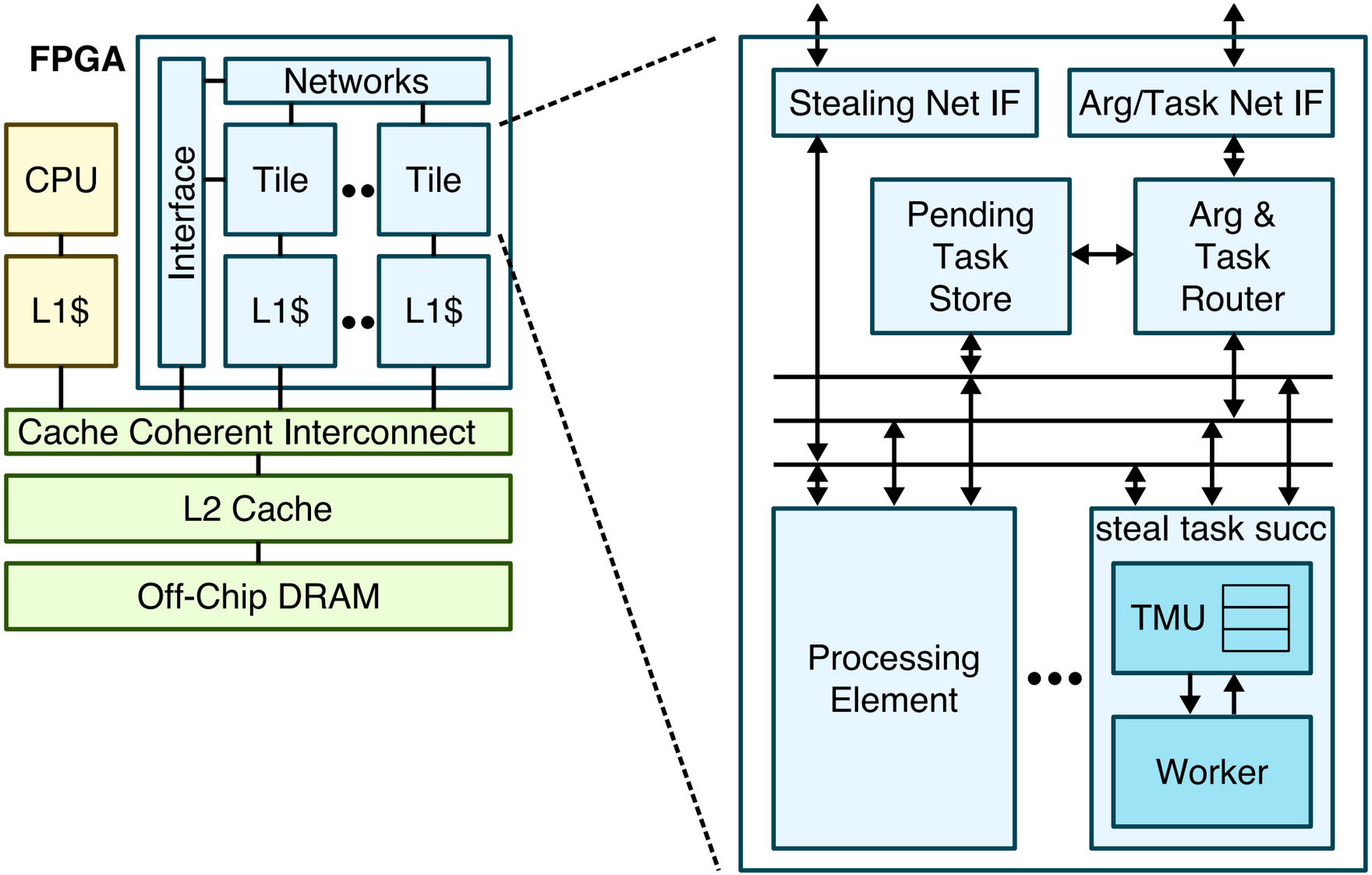
- Work stealing has good performance, space requirements, and communication overheads in both theory and practice

Scheduling Tasks with Work Stealing

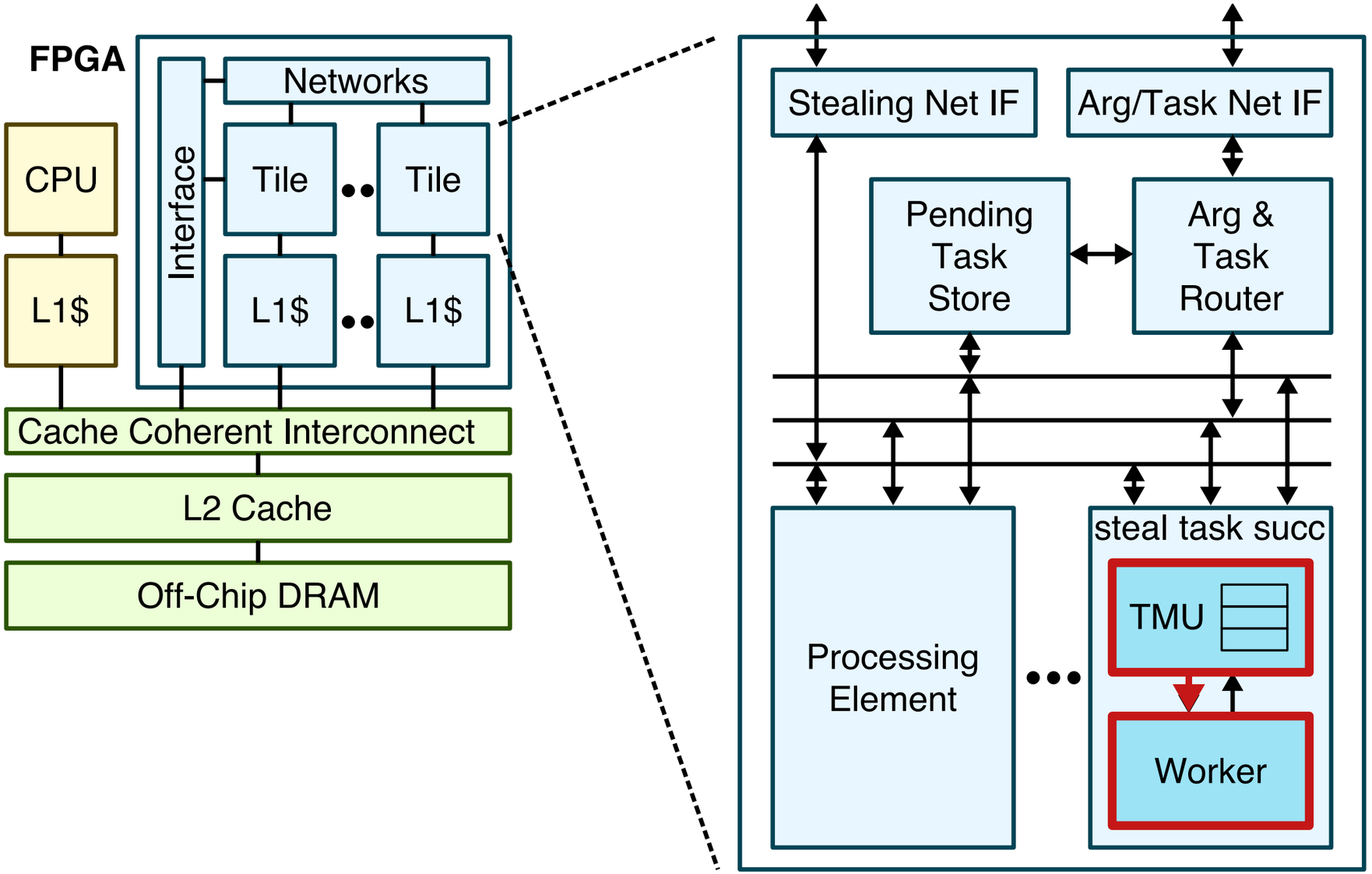


- Work stealing has good performance, space requirements, and communication overheads in both theory and practice

BanditXL Architectural Template

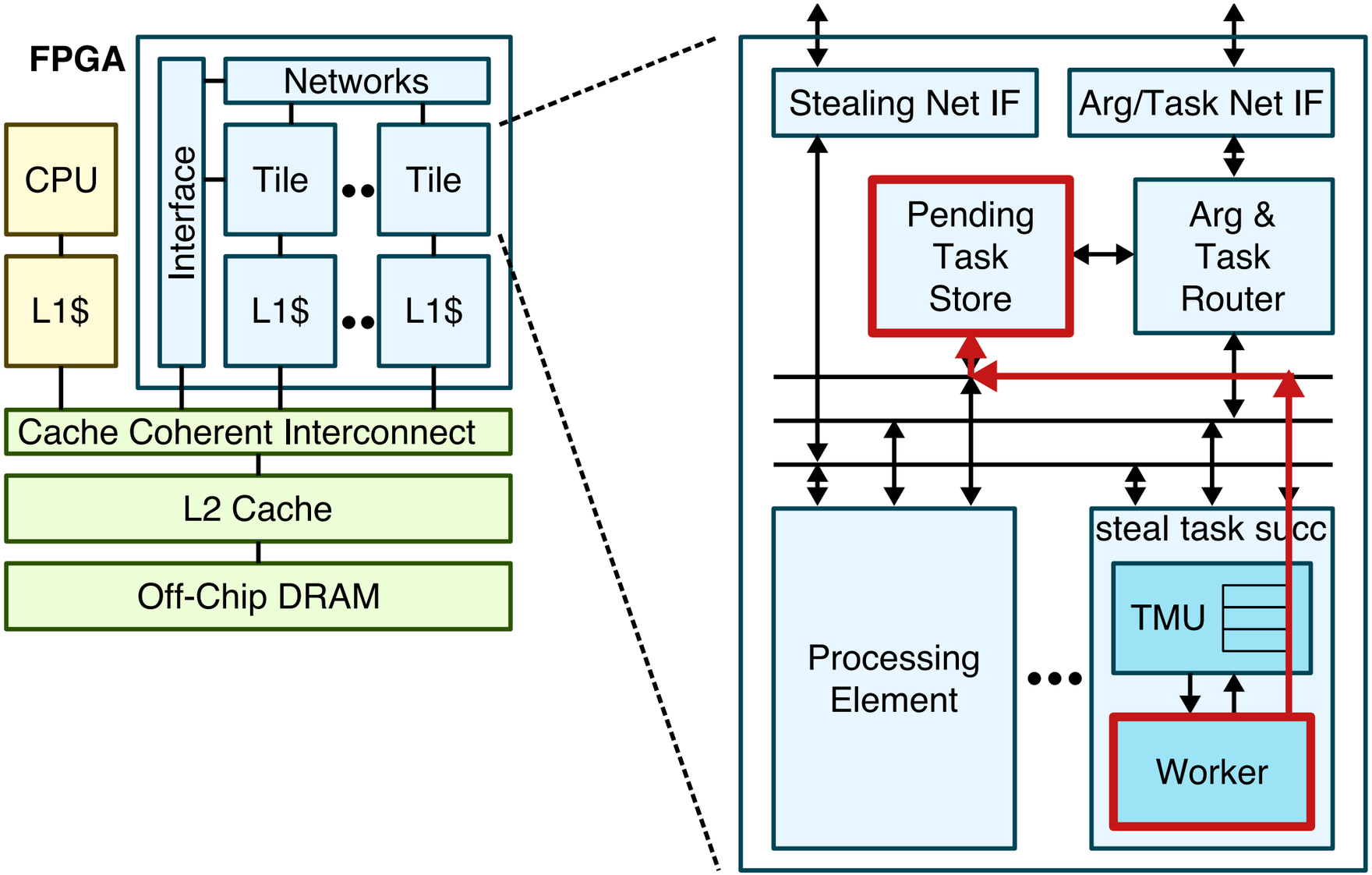


BanditXL Architectural Template



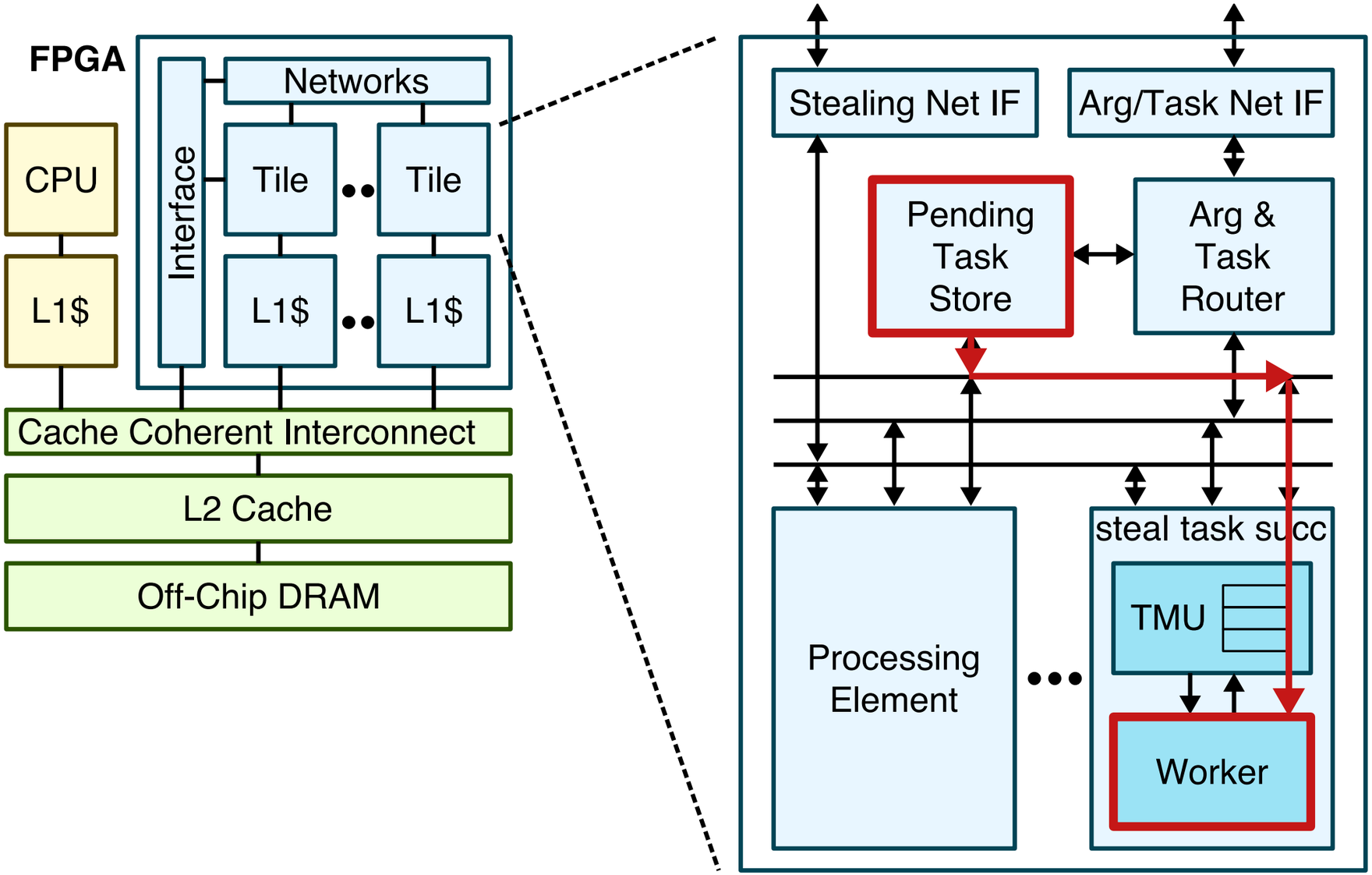
TMU dequeues task and sends to worker

BanditXL Architectural Template



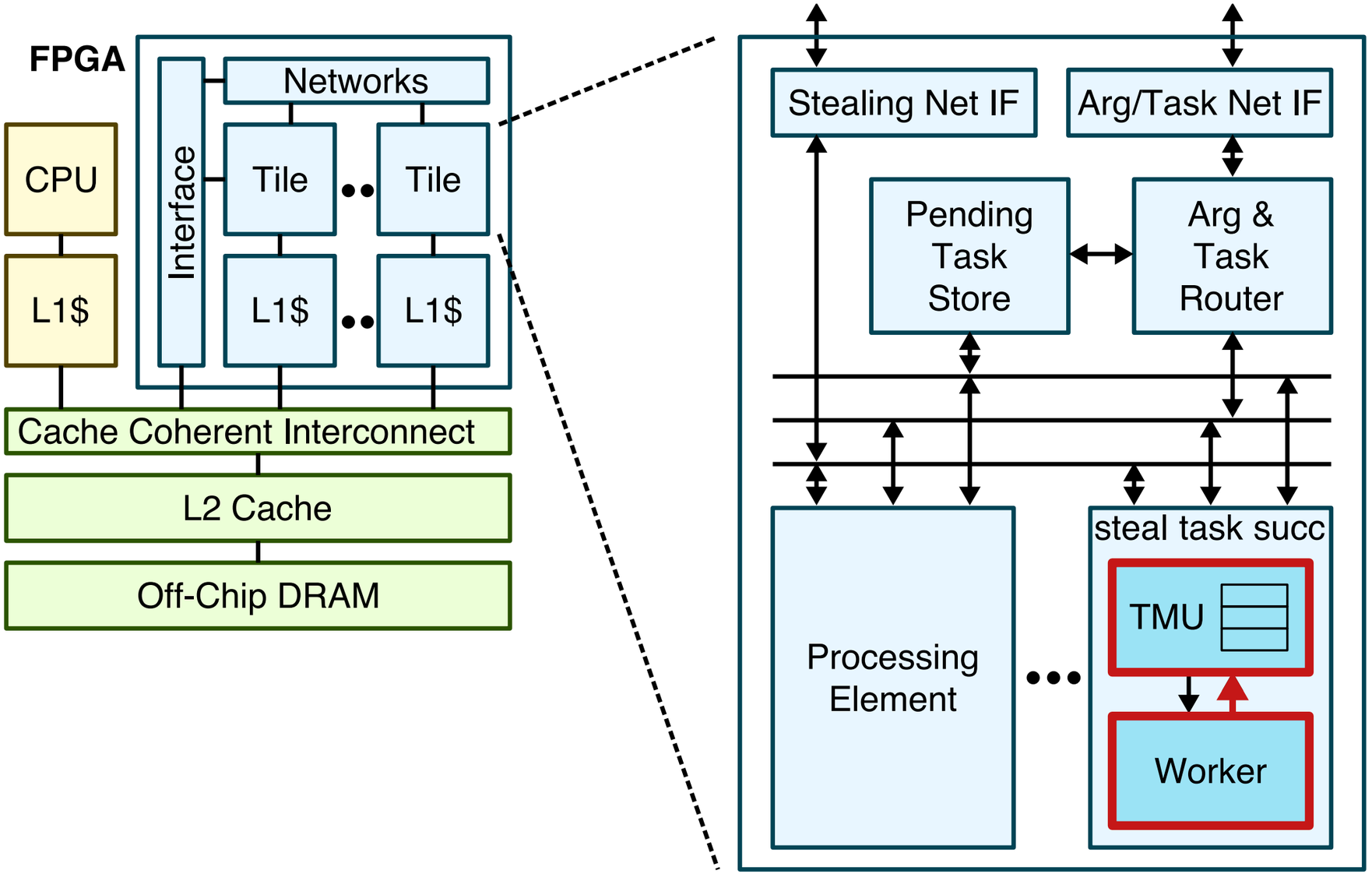
Worker sends request to pending task store to create a successor

BanditXL Architectural Template



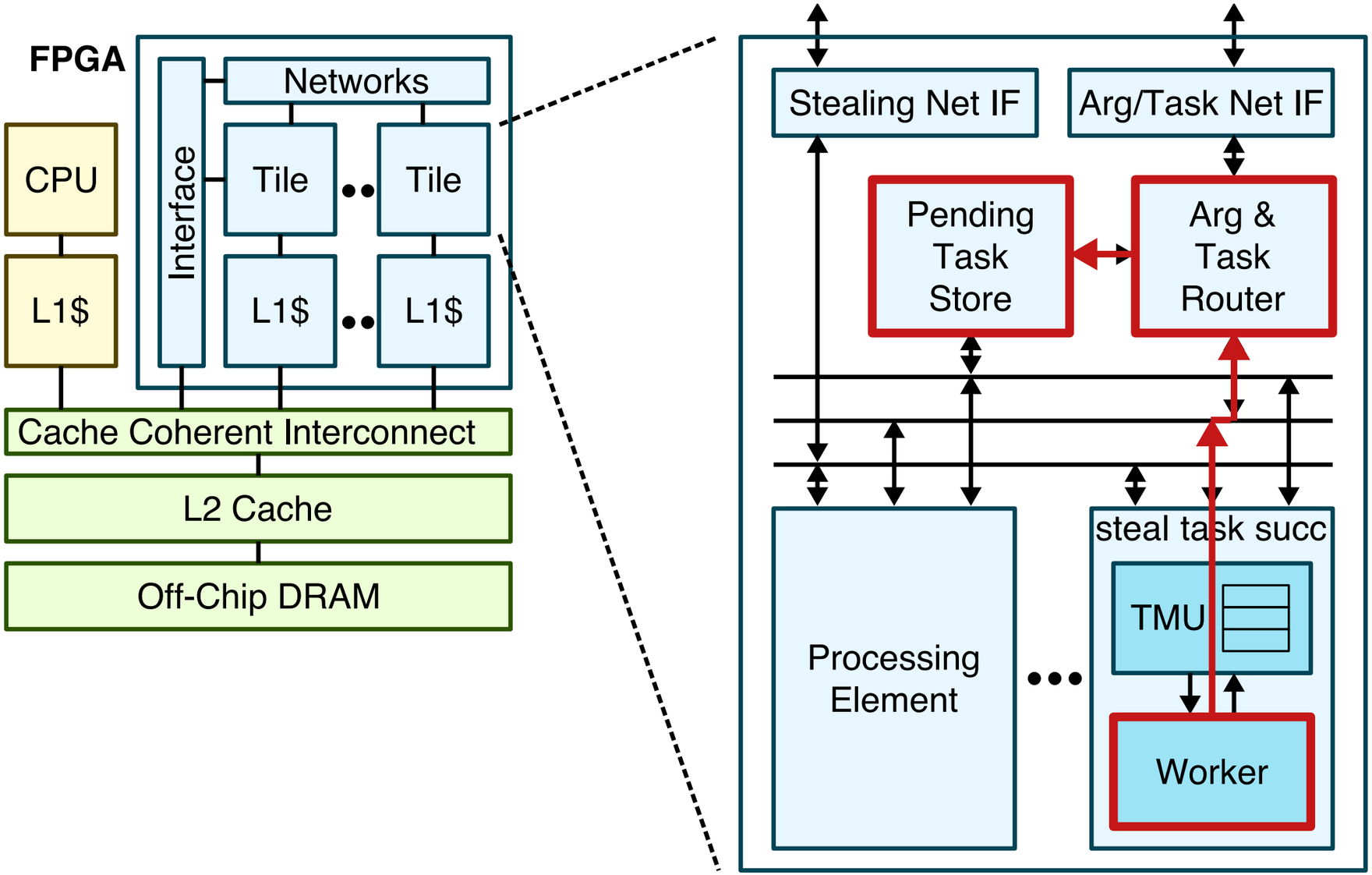
Pending task store sends worker response with ID for creating continuations

BanditXL Architectural Template



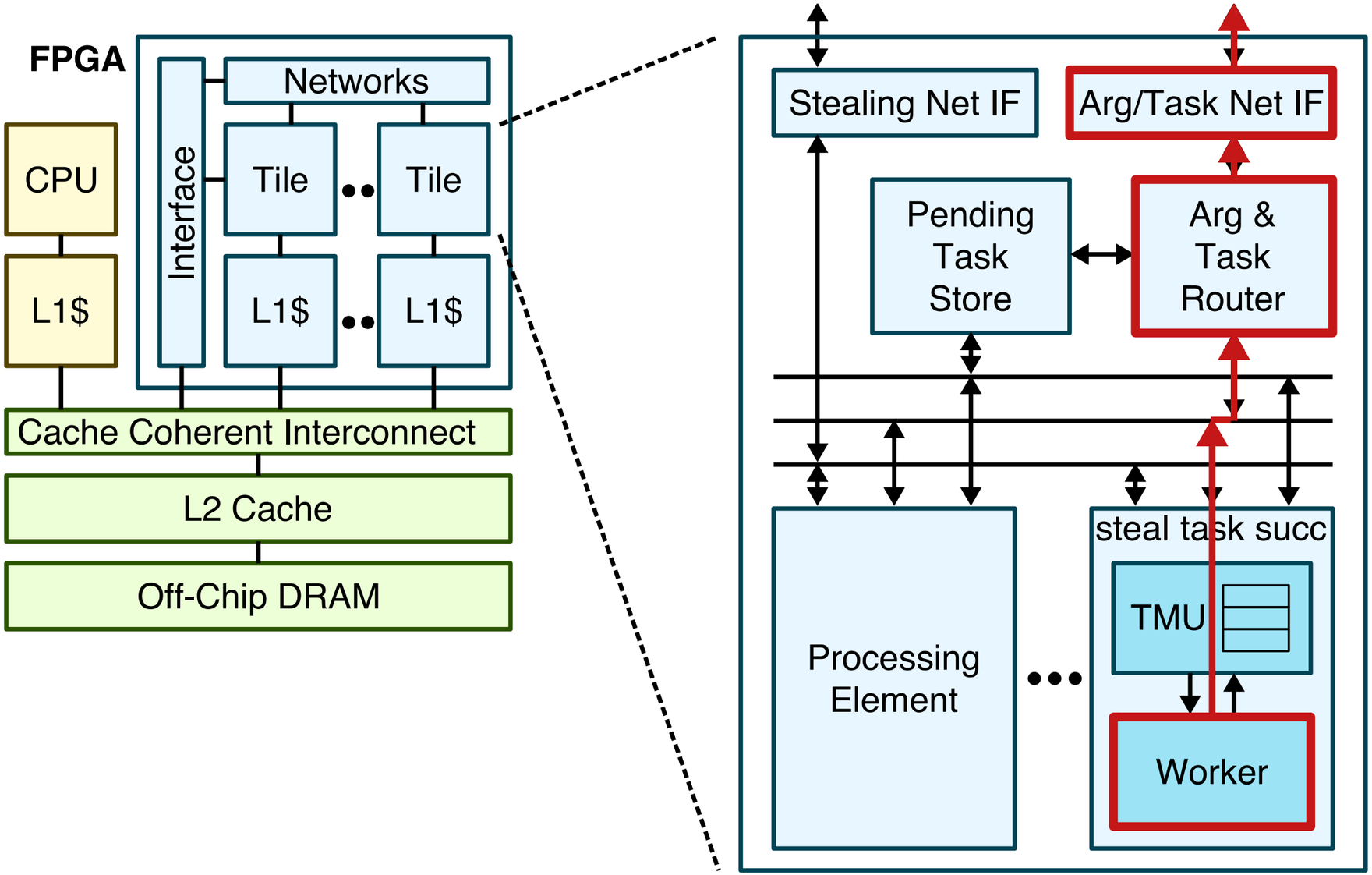
Worker sends spawed tasks to TMU

BanditXL Architectural Template



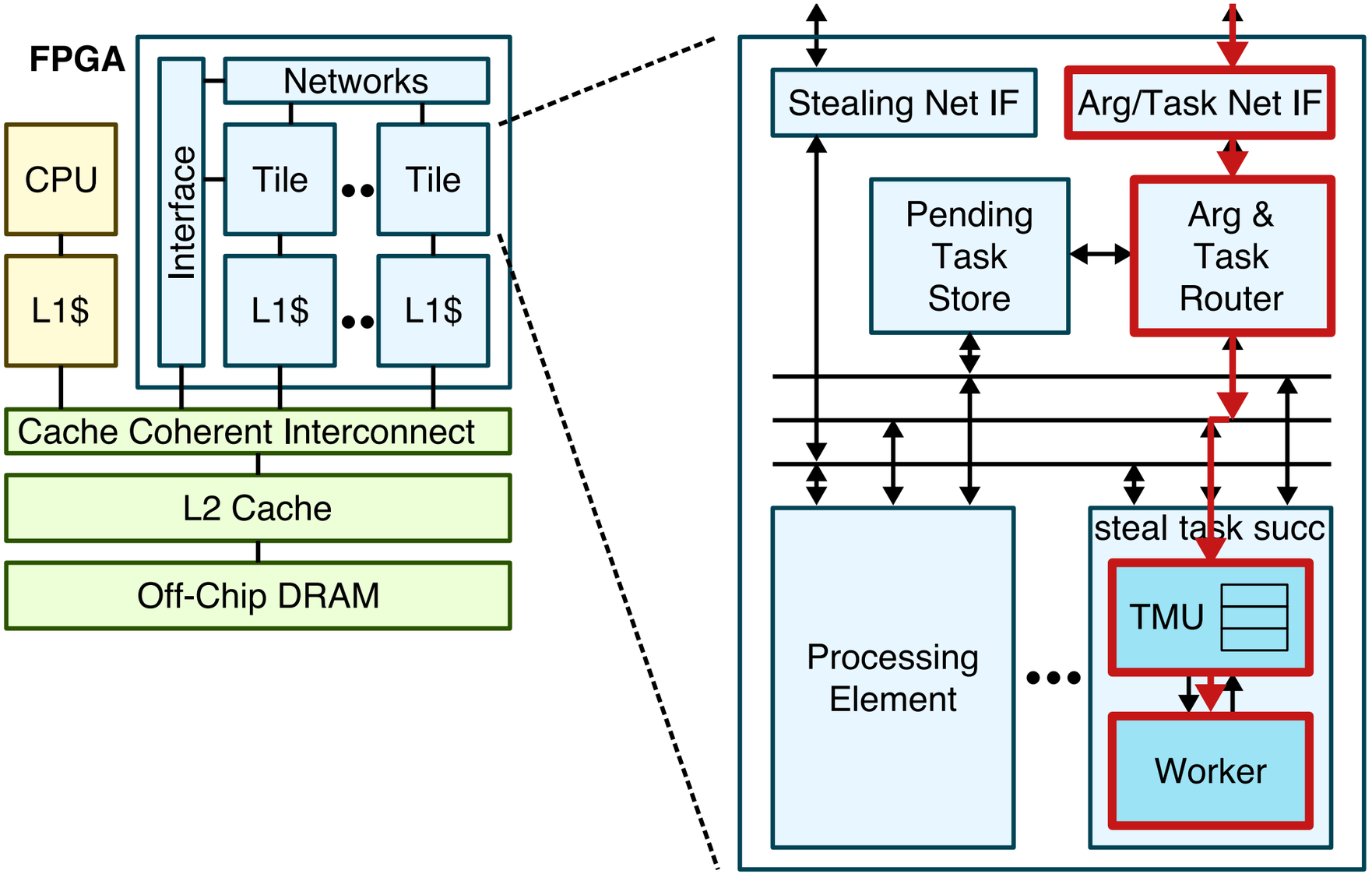
Worker sends return value to arg/task router

BanditXL Architectural Template



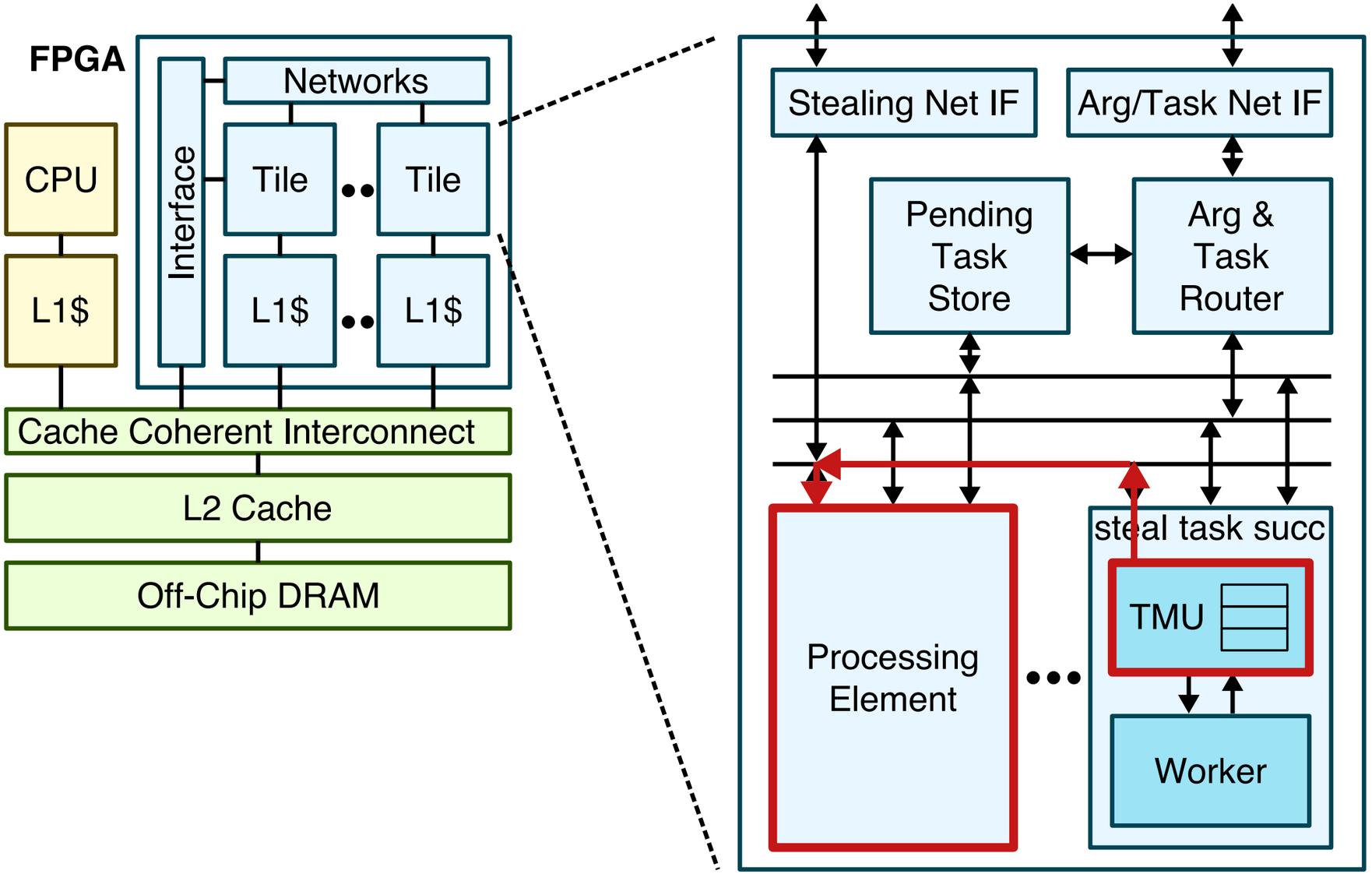
Worker sends return value to arg/task router

BanditXL Architectural Template



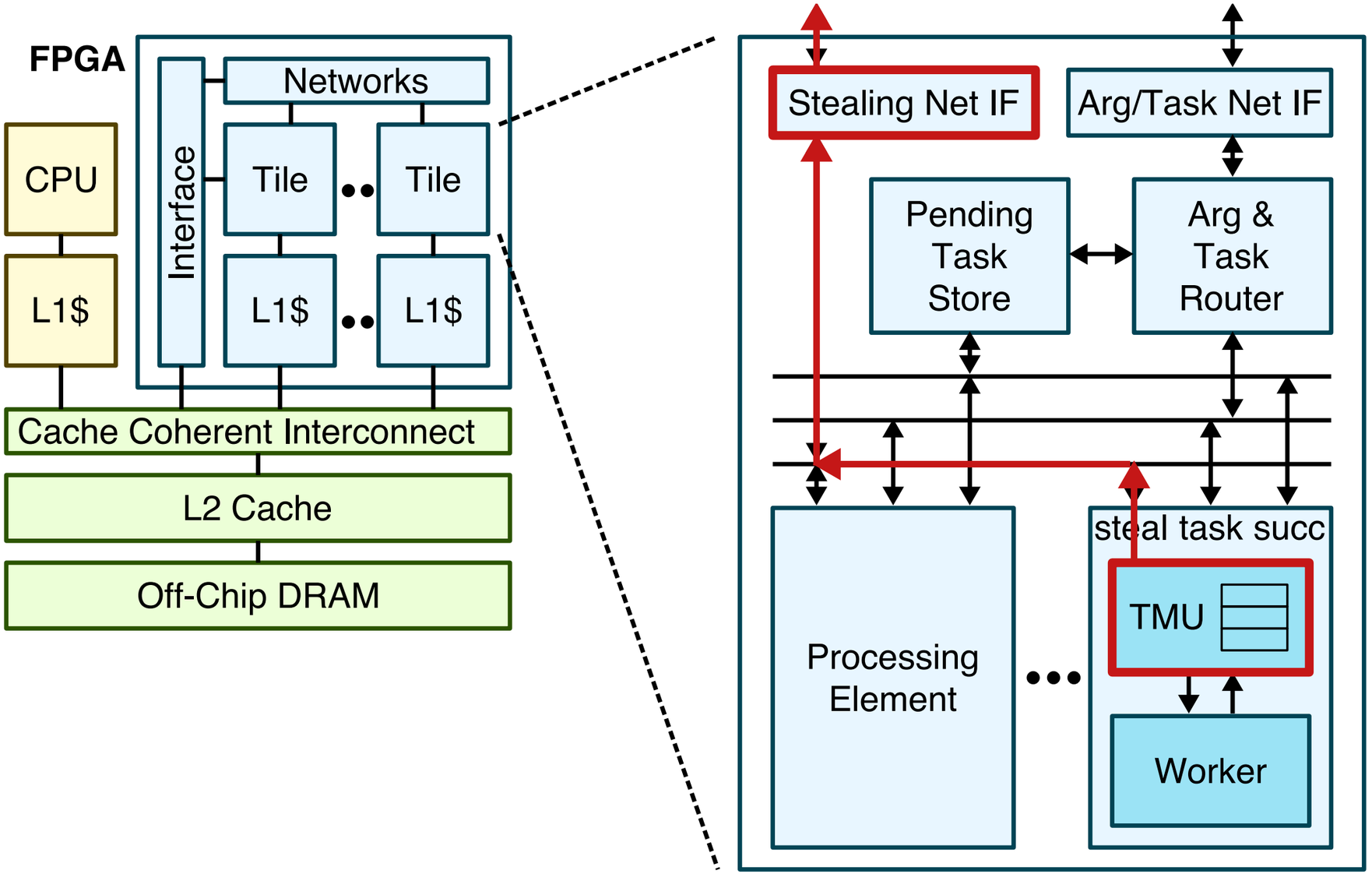
If this is the final argument pending task store sends now ready task back to TMU

BanditXL Architectural Template



If task queue is empty, TMU randomly selects victim to steal from

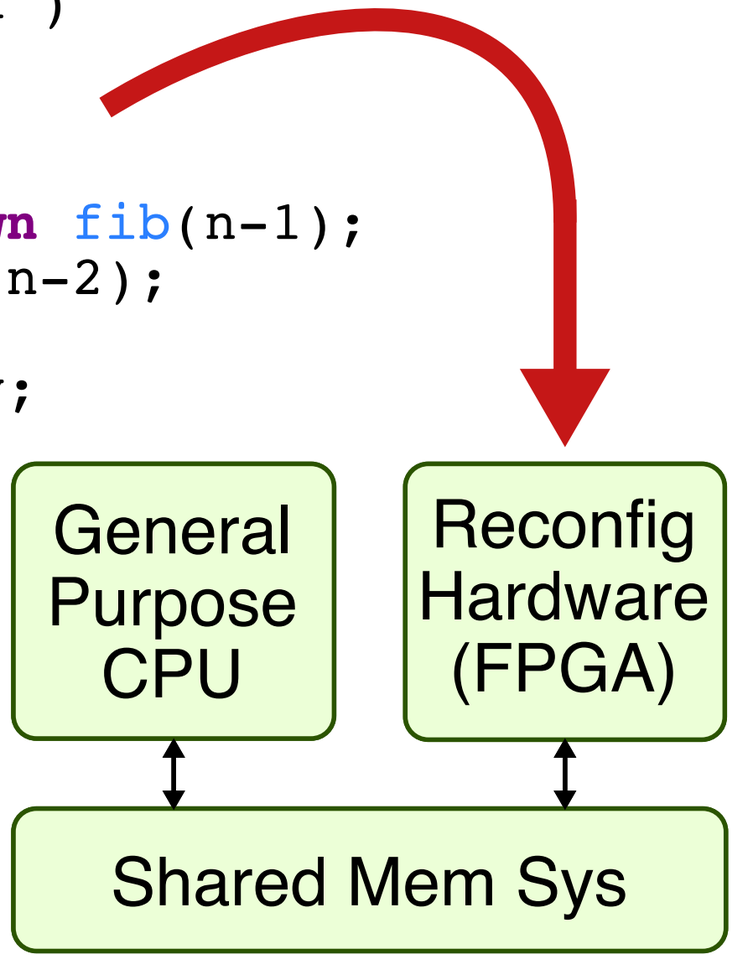
BanditXL Architectural Template



If task queue is empty, TMU randomly selects victim to steal from

BanditXL: Accelerating *Dynamic* Parallel Algorithms on Reconfigurable Hardware

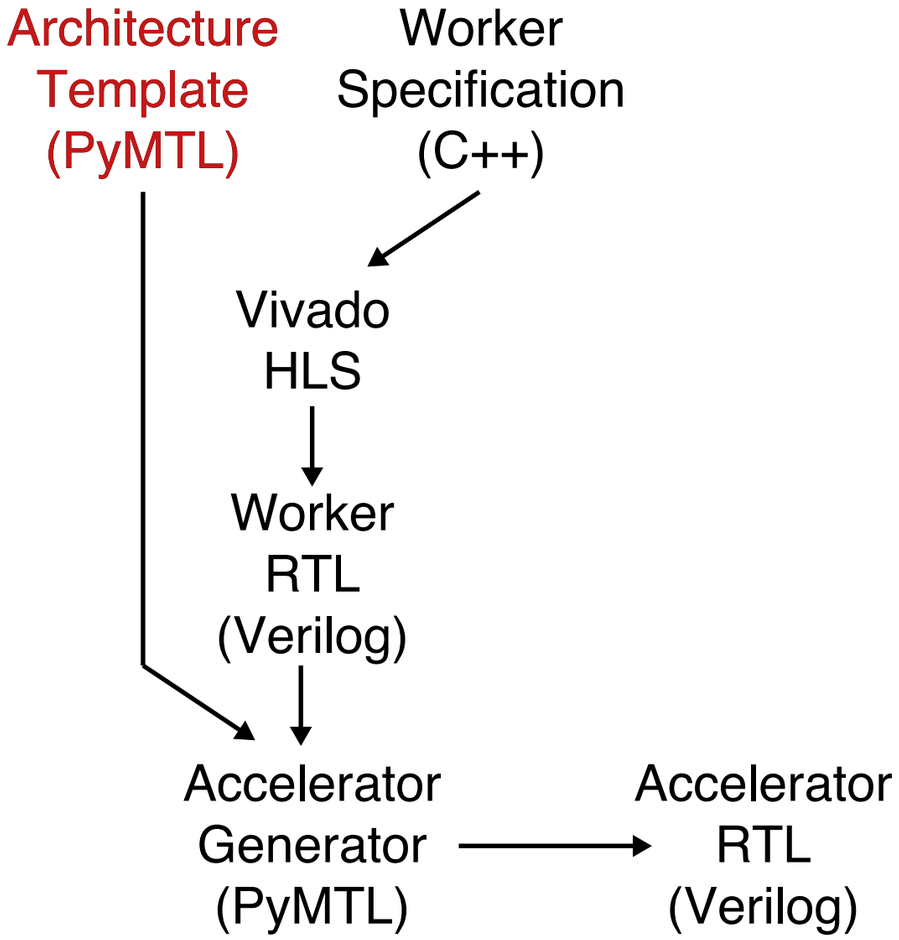
```
int fib( int n )
{
  if (n < 2)
    return n;
  int x = spawn fib(n-1);
  int y = fib(n-2);
  sync;
  return x + y;
}
```

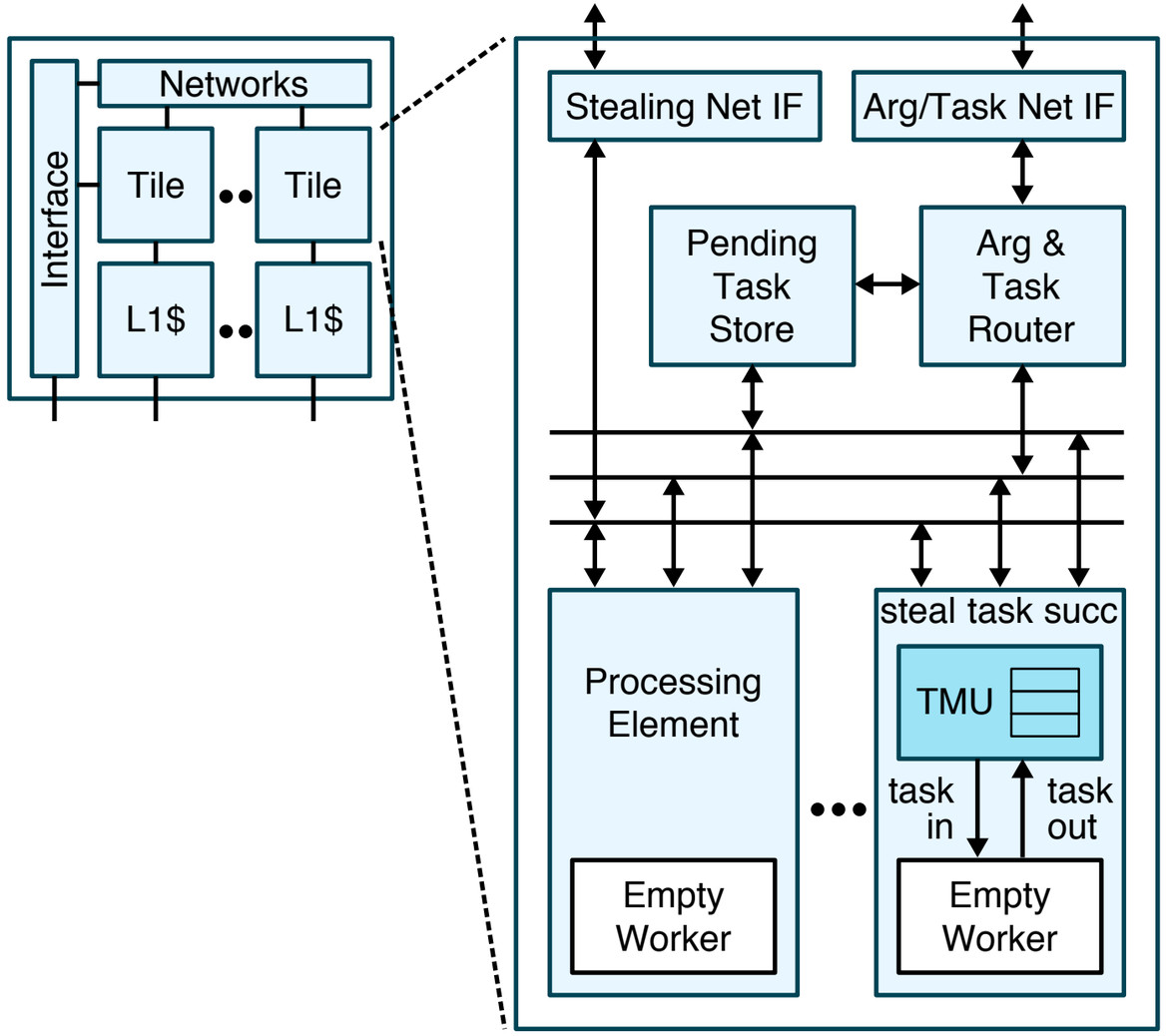


Tao Chen
Shreesha Srinath
Christopher Batten
G. Edward Suh
MICRO'18

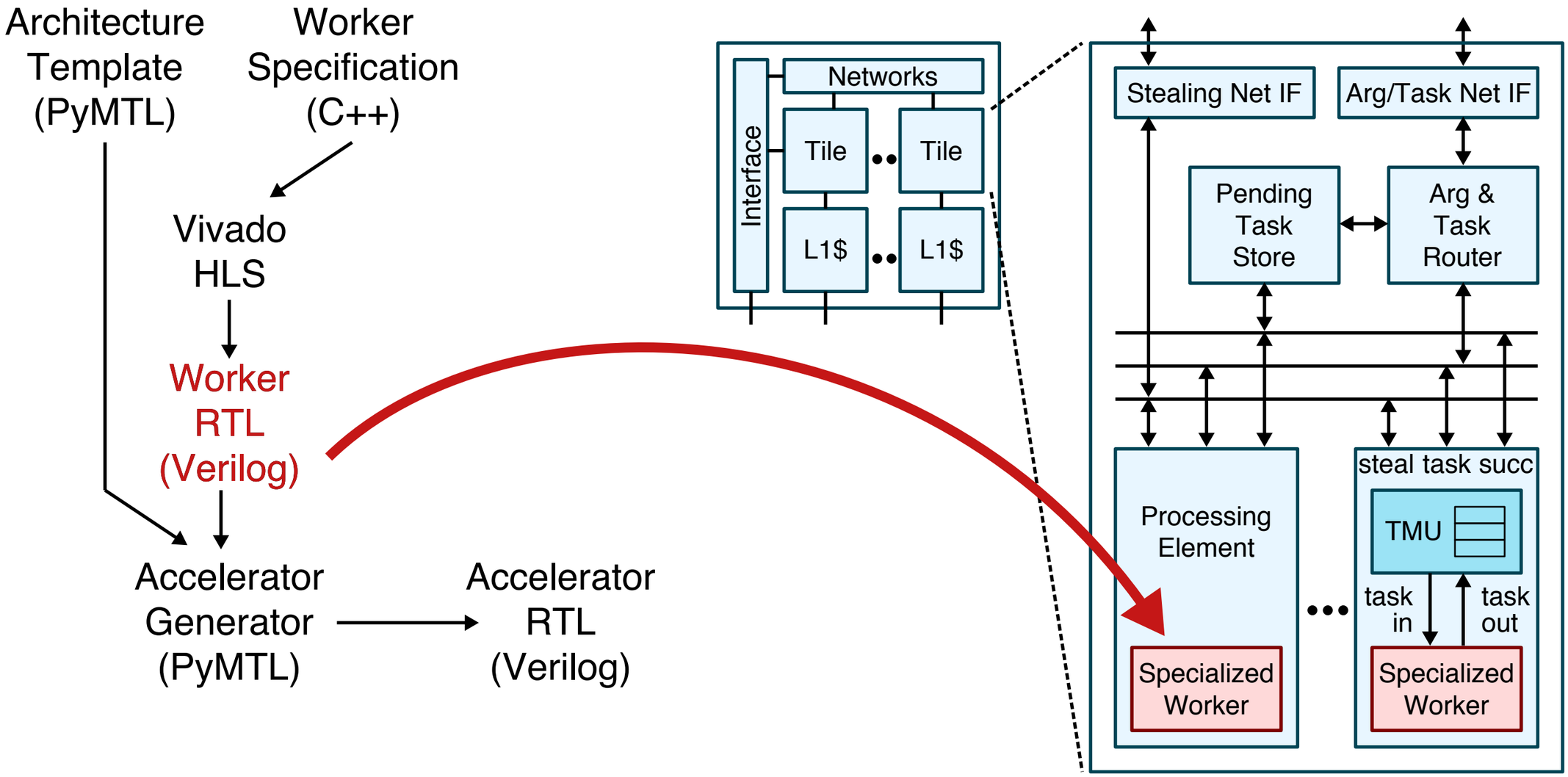
Motivation
Computation Model
Accelerator Architecture
Design Methodology
Evaluation
Ongoing Work

BanditXL Design Methodology

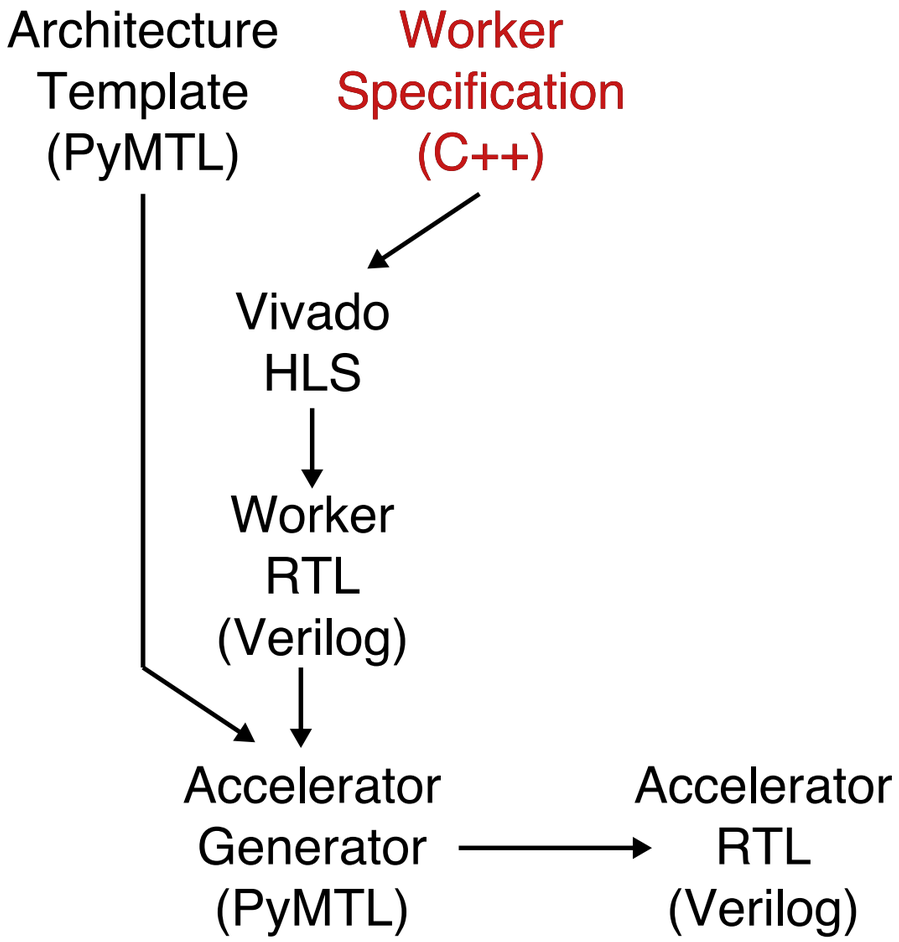




BanditXL Design Methodology

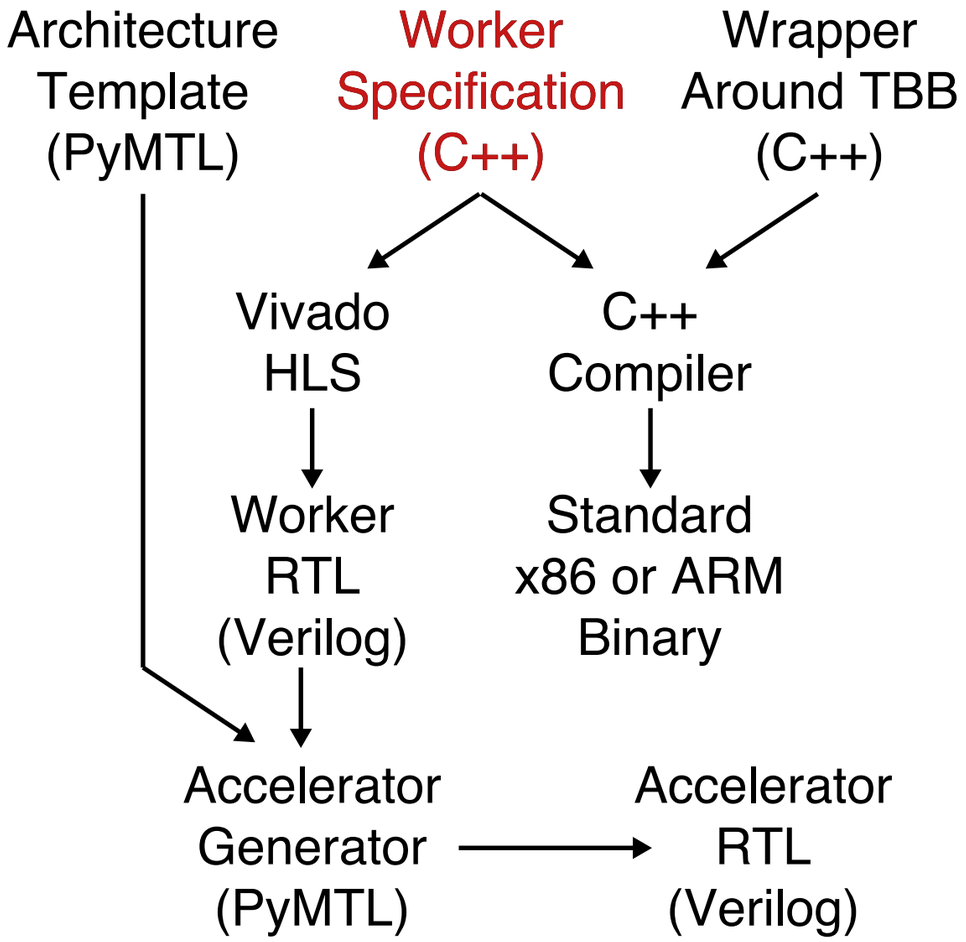


BanditXL Design Methodology




```
void FibWorkerHLS(
  TaskInPort<FibTask> tin,  TaskOutPort<FibTask> tout,
  SuccReqPort         sreq, SuccRespPort         sresp,
  ArgOutPort          aout )
{
  FibTask  task = task_in.read();
  task_k_t k    = task.k;
  if (task.type == FIB) {
    int n = task.x;
    if (n < 2)
      send_arg( Arg(k,n), aout );
    else {
      k = make_succ( SUM, k, 2, sreq, sresp );
      spawn( FibTask( FIB, k, 1, n-2), tout );
      spawn( FibTask( FIB, k, 0, n-1), tout );
    }
  }
  else if (task.type == SUM) {
    int sum = task.x + task.y;
    send_arg(Arg(k, sum), aout);
  }
}
```

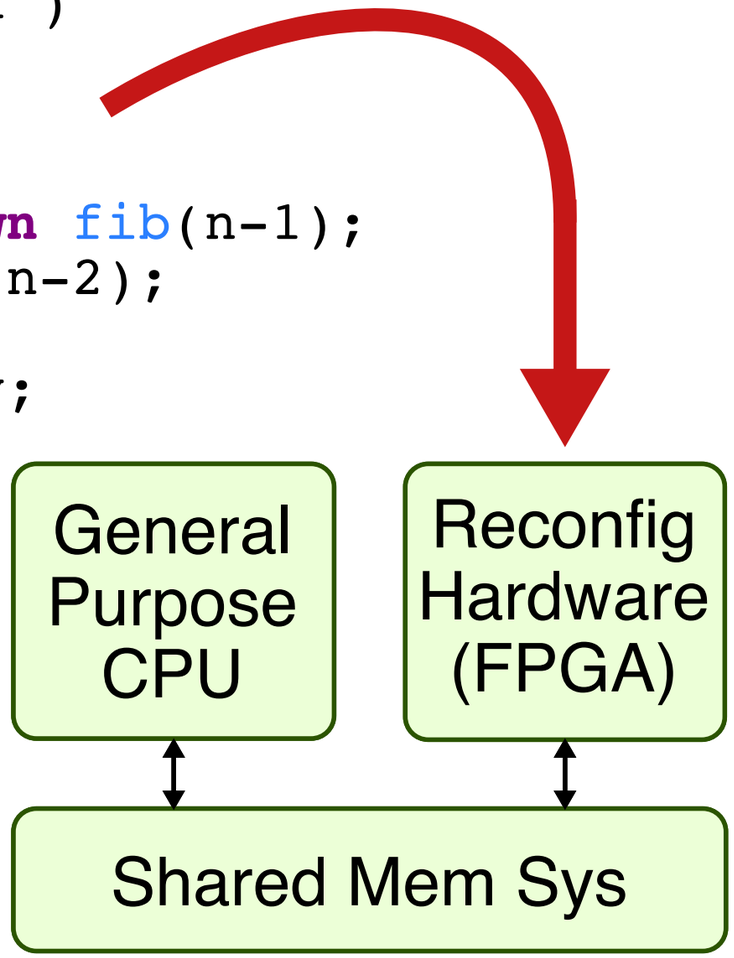
BanditXL Design Methodology




```
void FibWorkerHLS(
  TaskInPort<FibTask> tin,  TaskOutPort<FibTask> tout,
  SuccReqPort         sreq, SuccRespPort         sresp,
  ArgOutPort          aout )
{
  FibTask  task = task_in.read();
  task_k_t k    = task.k;
  if (task.type == FIB) {
    int n = task.x;
    if (n < 2)
      send_arg( Arg(k,n), aout );
    else {
      k = make_succ( SUM, k, 2, sreq, sresp );
      spawn( FibTask( FIB, k, 1, n-2), tout );
      spawn( FibTask( FIB, k, 0, n-1), tout );
    }
  }
  else if (task.type == SUM) {
    int sum = task.x + task.y;
    send_arg(Arg(k, sum), aout);
  }
}
```

BanditXL: Accelerating *Dynamic* Parallel Algorithms on Reconfigurable Hardware

```
int fib( int n )
{
  if (n < 2)
    return n;
  int x = spawn fib(n-1);
  int y = fib(n-2);
  sync;
  return x + y;
}
```



Tao Chen
Shreesha Srinath
Christopher Batten
G. Edward Suh
MICRO'18

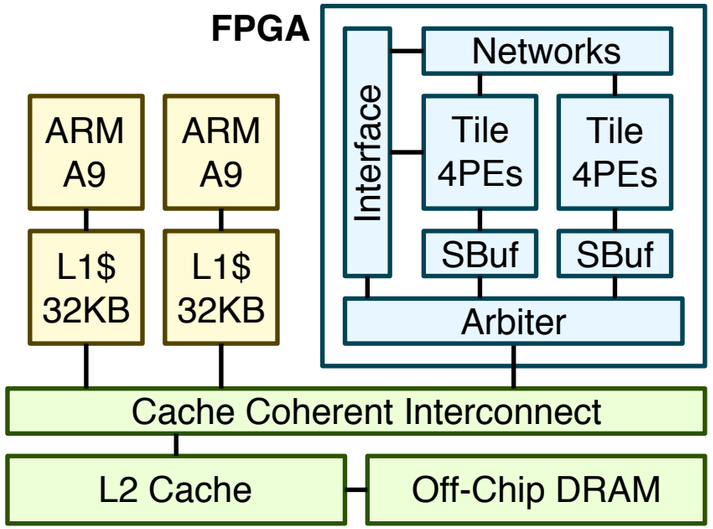
Motivation
Computation Model
Accelerator Architecture
Design Methodology
Evaluation
Ongoing Work

Application Kernels

Name	Suite	Description	Pattern
nw	in-house	Needleman-Wunsch Algorithm	data-flow
quicksort	in-house	quicksort algorithm	fork/join
cilksort	Cilk apps	parallel merge sort algorithm	fork/join
queens	Cilk apps	N-queens problem	fork/join
knapsack	Cilk apps	0-1 knapsack problem	fork/join
uts	UTS	unbalanced tree search	fork/join
bbgemm	MachSuite	blocked matrix multiplication	data-parallel
bfsqueue	MachSuite	breadth first search	data-parallel
spmvcrs	MachSuite	sparse matrix-vector mult	data-parallel
stencil2d	MachSuite	3D stencil computation	data-parallel

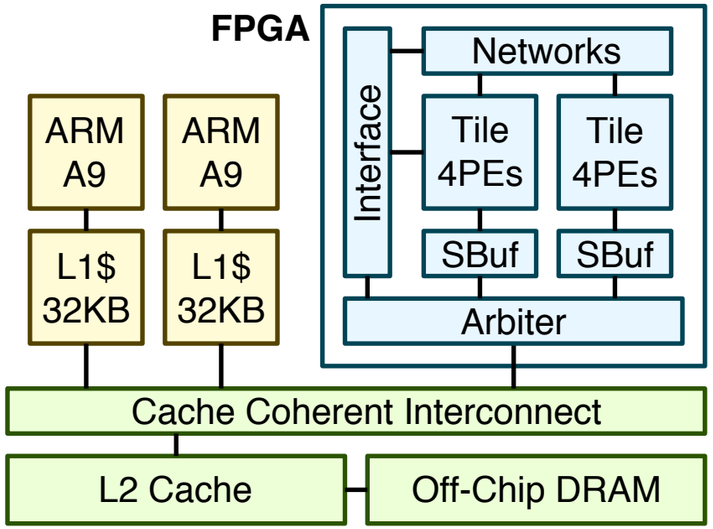
- Optimized software baseline using Intel Cilk Plus w/ ARM NEON auto-vectorization
- C++ application driver and worker implemented with design methodology

Current and Future CPU+FPGA Platforms

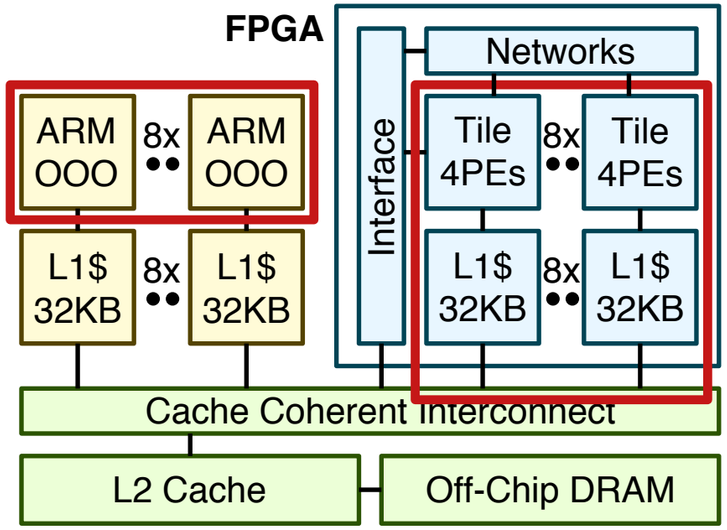


- **Current Zynq-7000 SoC Platform**
  - Prototype using Zedboard
  - Two 667MHz ARM Cortex-A9 cores
  - Xilinx 7-series integrated FPGA fabric (modest capacity, 142MHz)
  - Accelerator uses stream buffers
  - Lower BW: FPGA $\leftrightarrow$ coherent mem sys

Current and Future CPU+FPGA Platforms

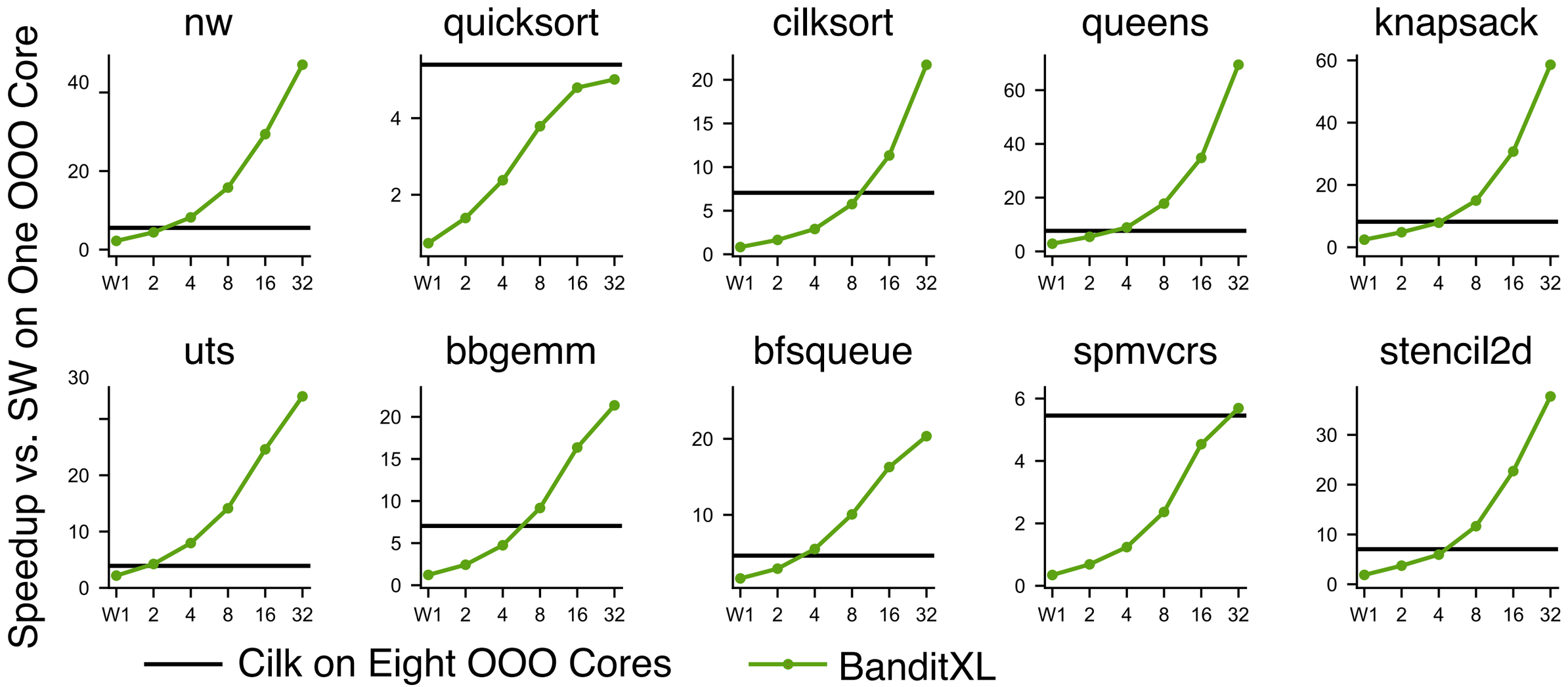


- **Current Zynq-7000 SoC Platform**
  - Prototype using Zedboard
  - Two 667MHz ARM Cortex-A9 cores
  - Xilinx 7-series integrated FPGA fabric (modest capacity, 142MHz)
  - Accelerator uses stream buffers
  - Lower BW: FPGA $\leftrightarrow$ coherent mem sys



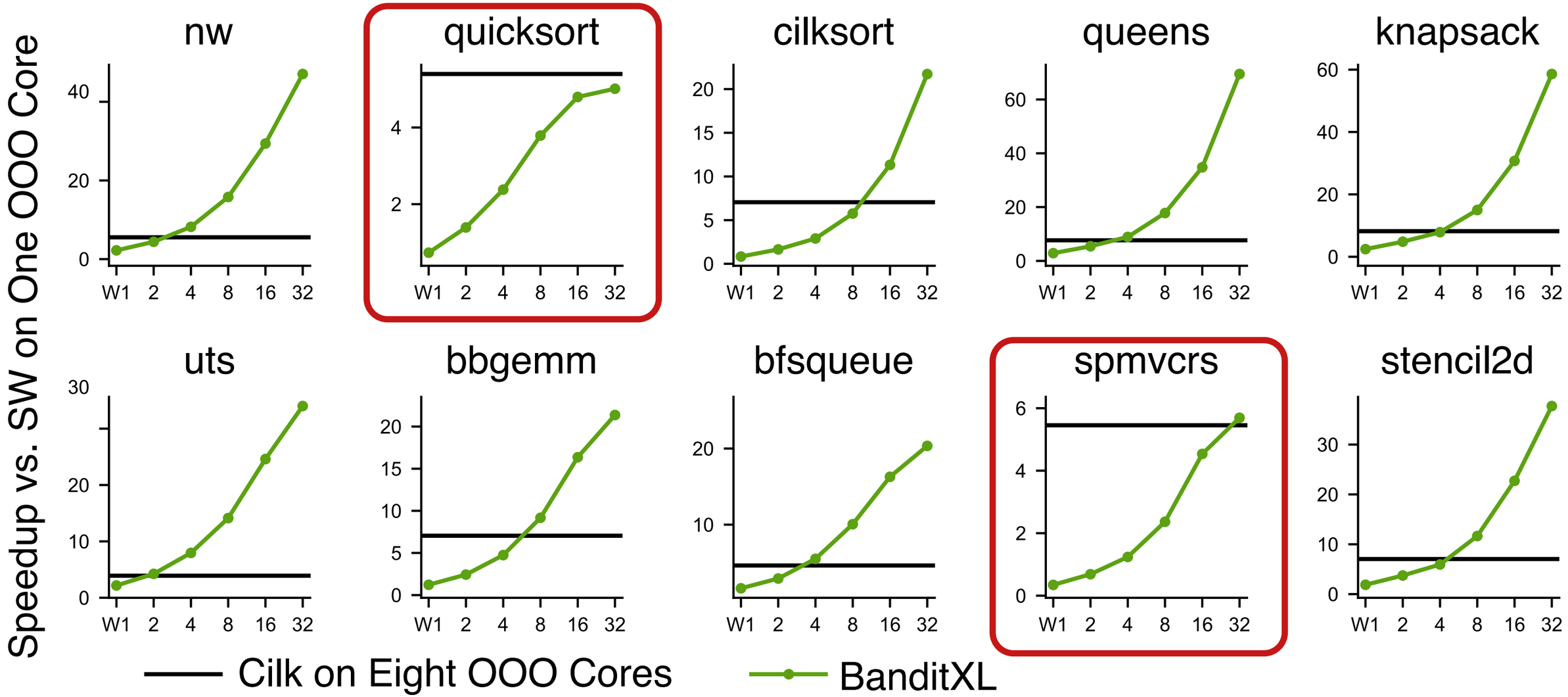
- **Future CPU+FPGA Platform**
  - Simulation study using gem5
  - Eight 1GHz ARM 4-way OOO cores
  - Xilinx 7-series integrated FPGA fabric (larger capacity, 200MHz)
  - Accelerators uses coherent 2x-pumped 32KB L1$
  - Higher BW: FPGA $\leftrightarrow$ coherent mem sys

Speedup of BanditXL on Future CPU+FPGA Platform



- FlexArch is 4× faster than 8 cores, 24× faster than 1 core (geo mean)

Speedup of BanditXL on Future CPU+FPGA Platform

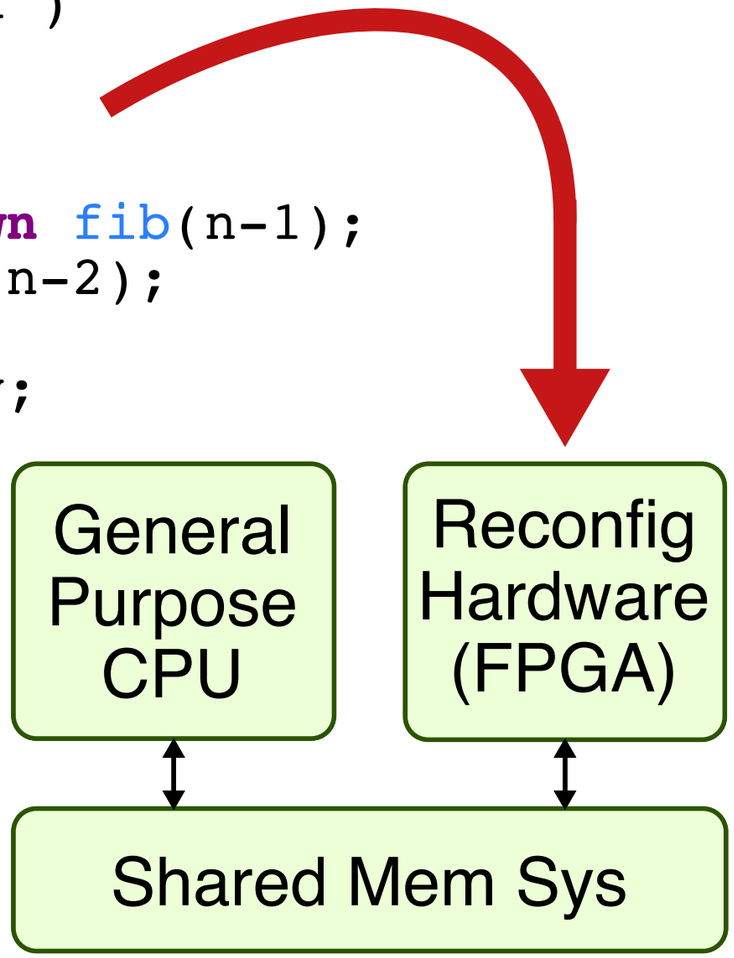


- FlexArch is 4× faster than 8 cores, 24× faster than 1 core (geo mean)

Applications Driving Architectures

BanditXL: Accelerating *Dynamic* Parallel Algorithms on Reconfigurable Hardware

```
int fib( int n )
{
  if (n < 2)
    return n;
  int x = spawn fib(n-1);
  int y = fib(n-2);
  sync;
  return x + y;
}
```



Tao Chen
Shreesha Srinath
Christopher Batten
G. Edward Suh
MICRO'18

Motivation

Computation Model

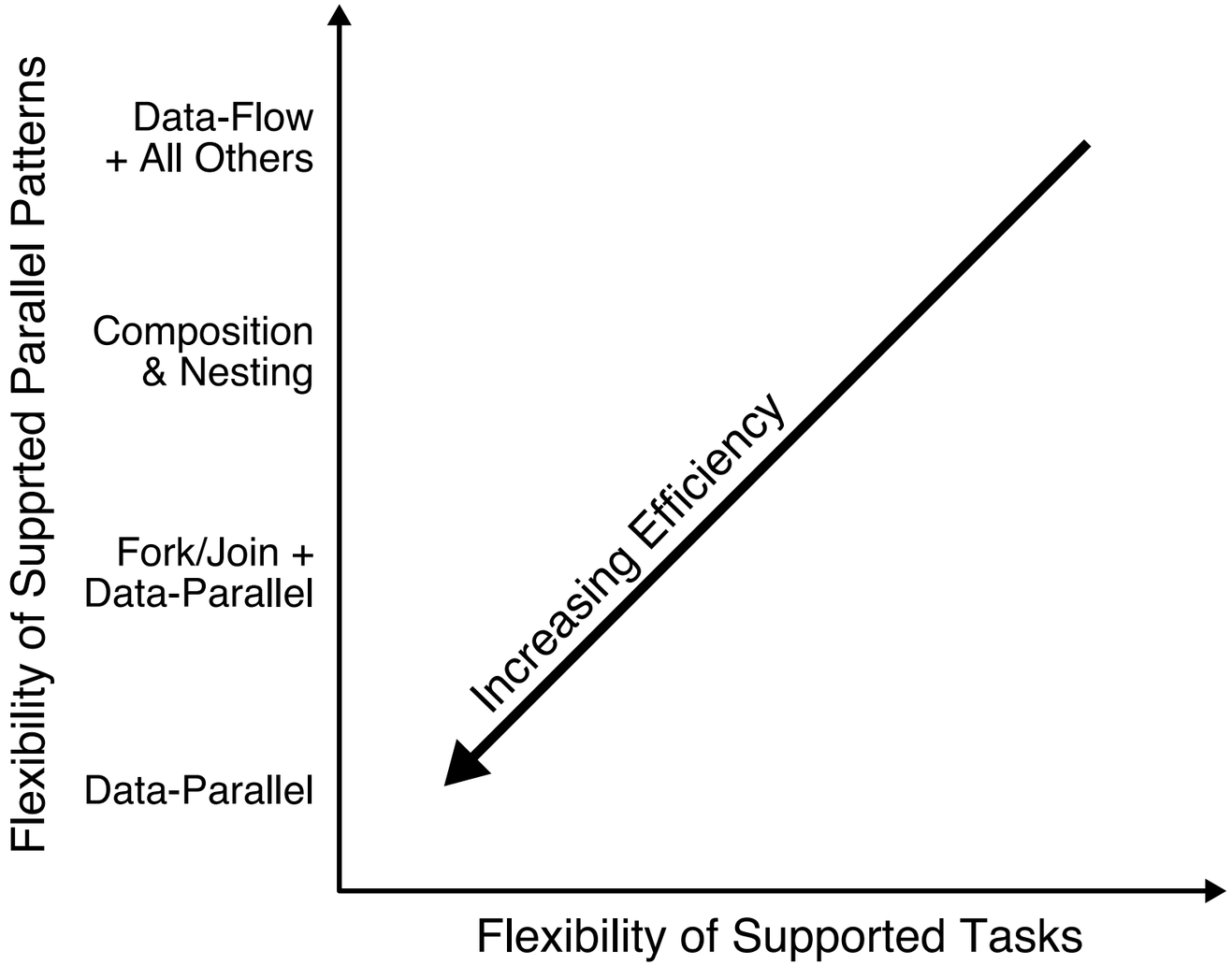
Accelerator Architecture

Design Methodology

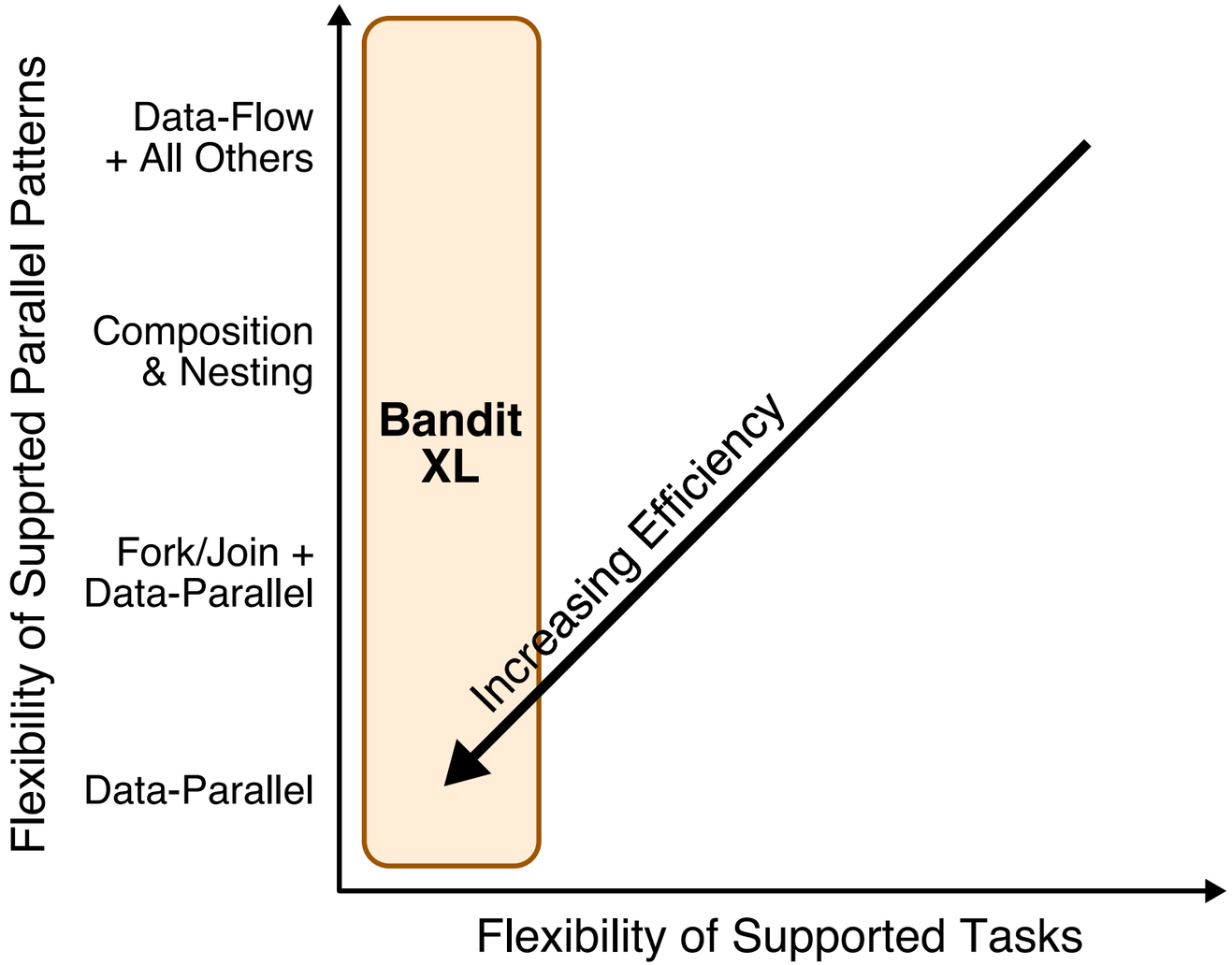
Evaluation

Ongoing Work

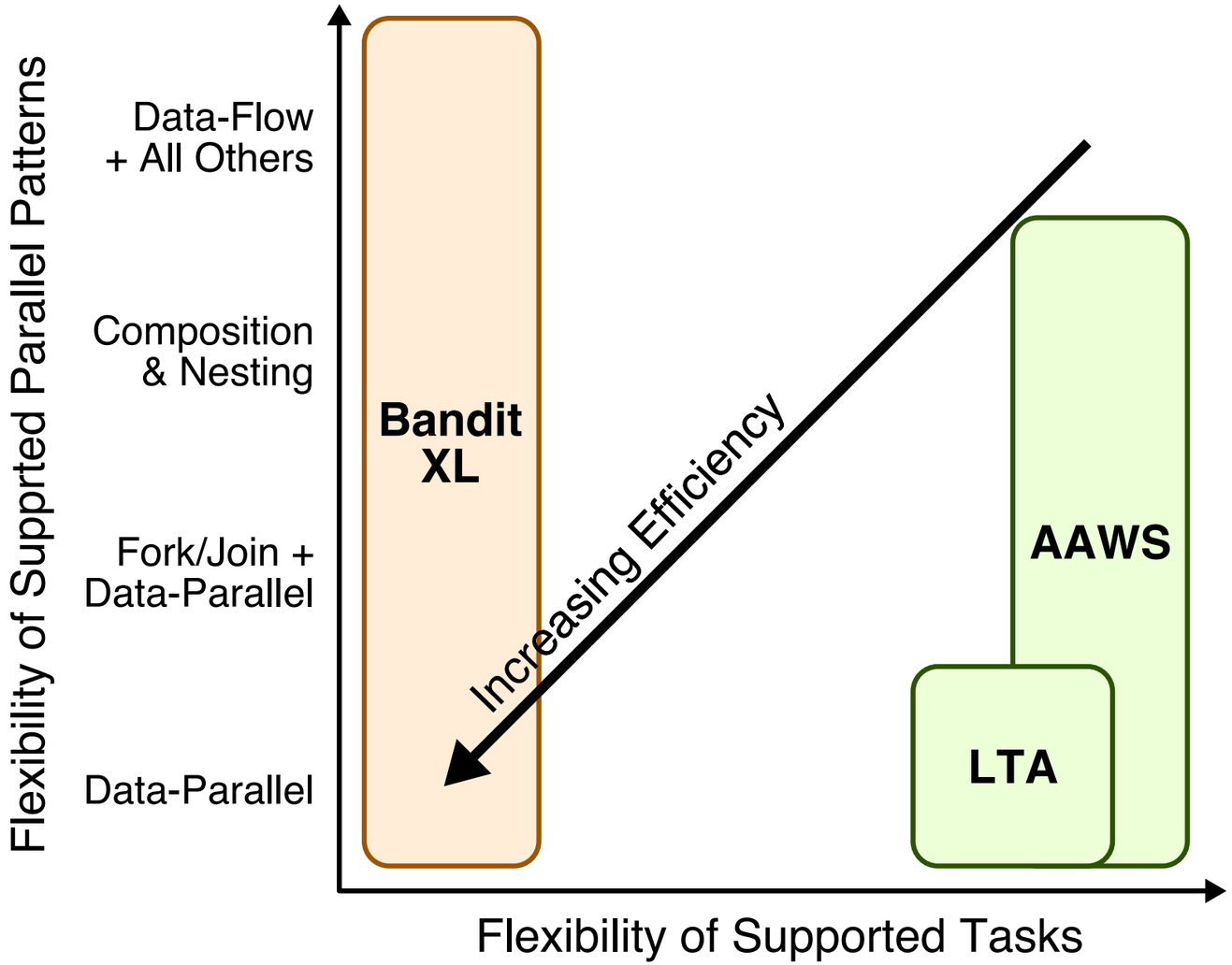
big.TINY: Balancing Flexibility and Efficiency



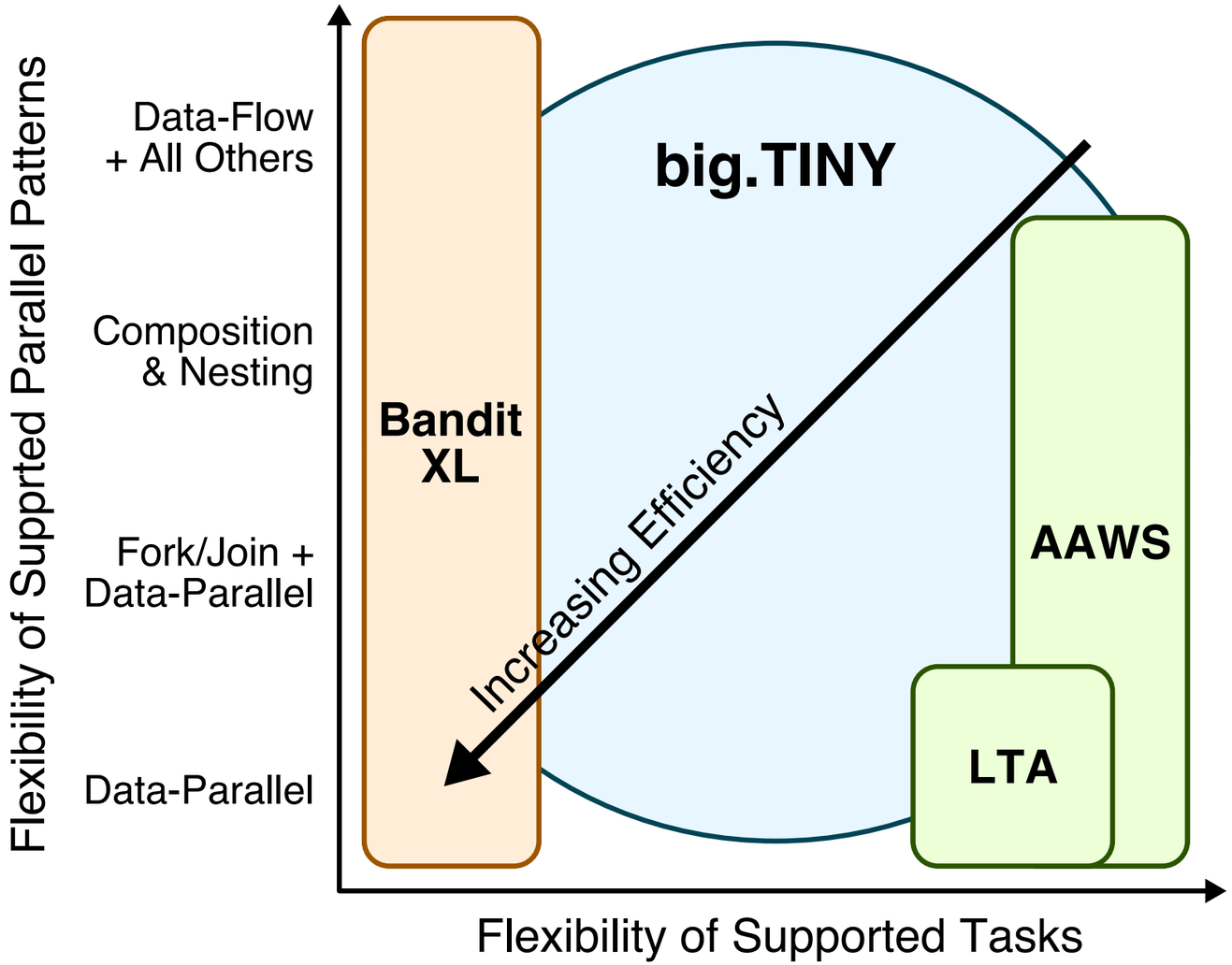
big.TINY: Balancing Flexibility and Efficiency



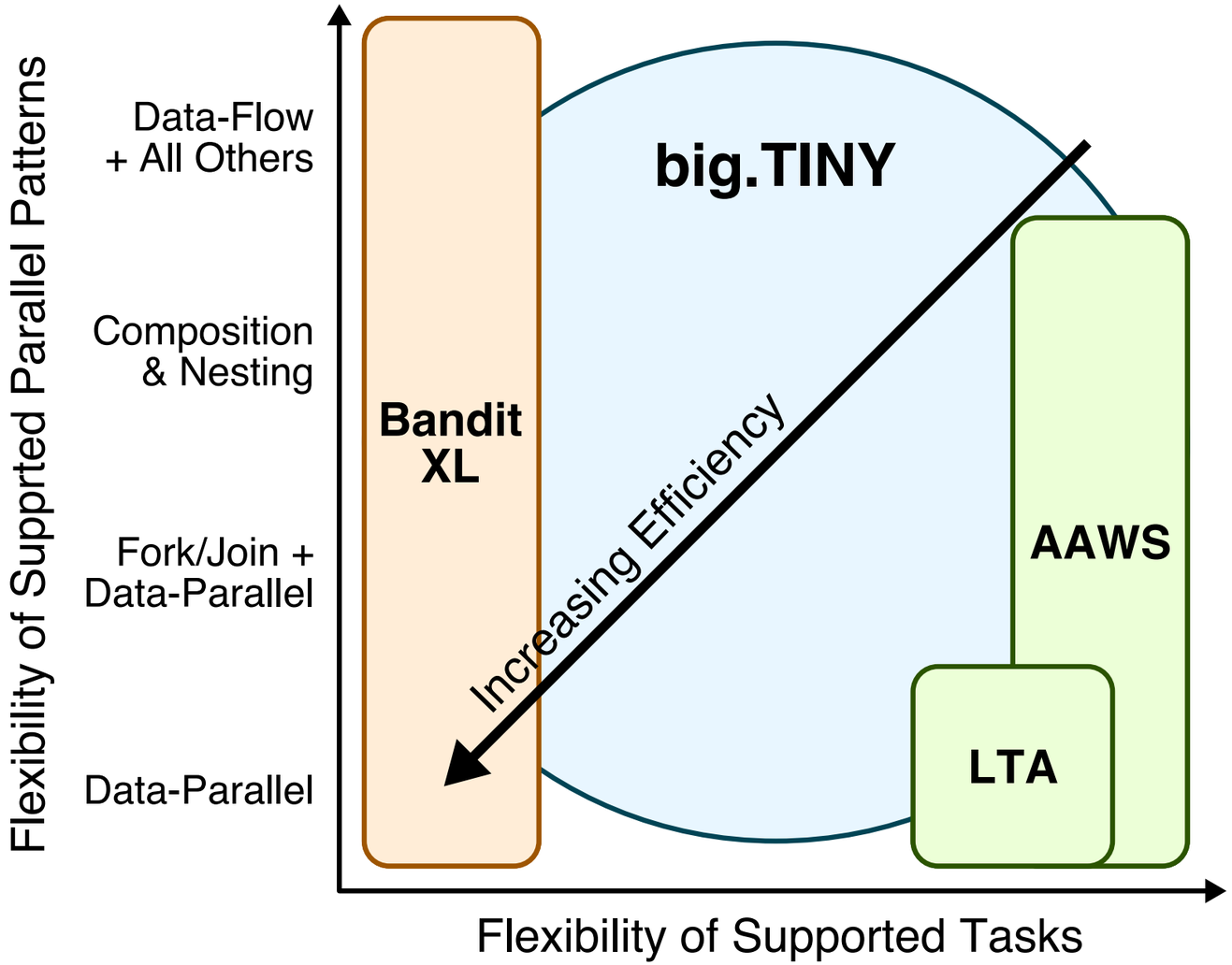
big.TINY: Balancing Flexibility and Efficiency

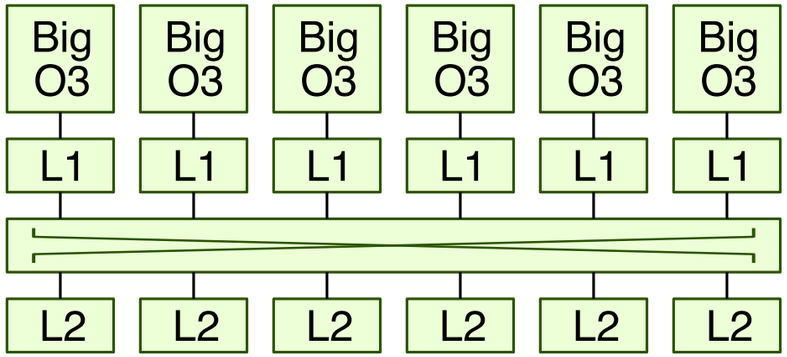


big.TINY: Balancing Flexibility and Efficiency

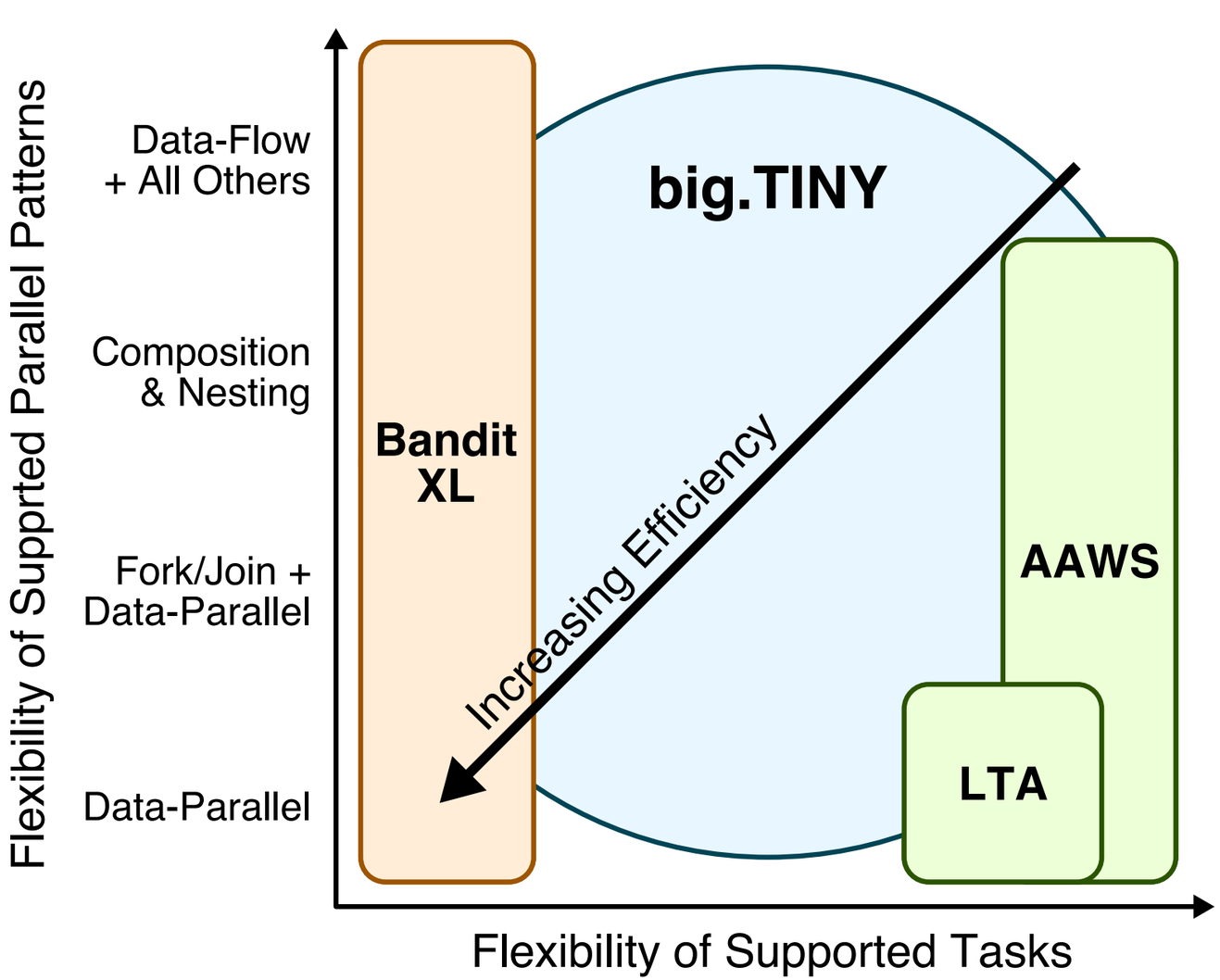


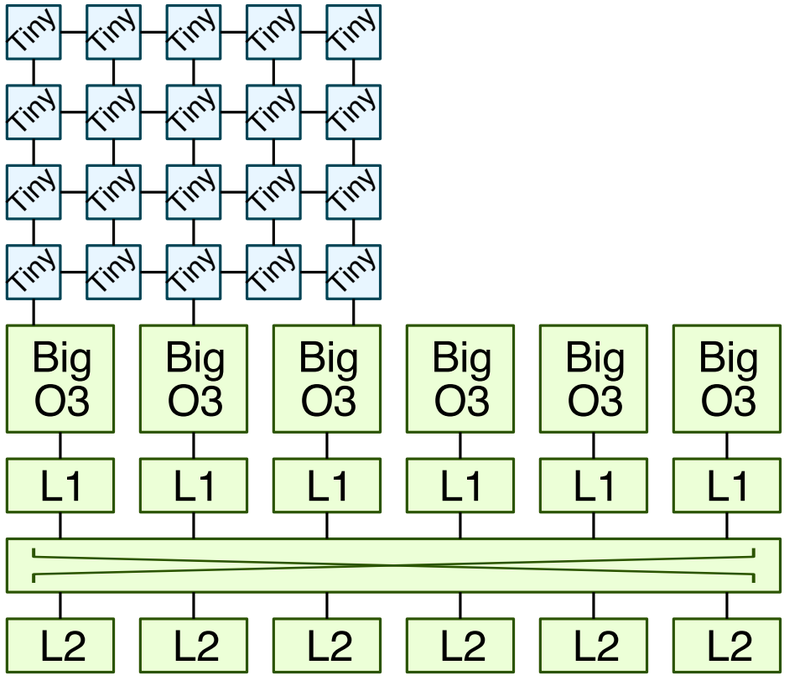
big.TINY: Balancing Flexibility and Efficiency



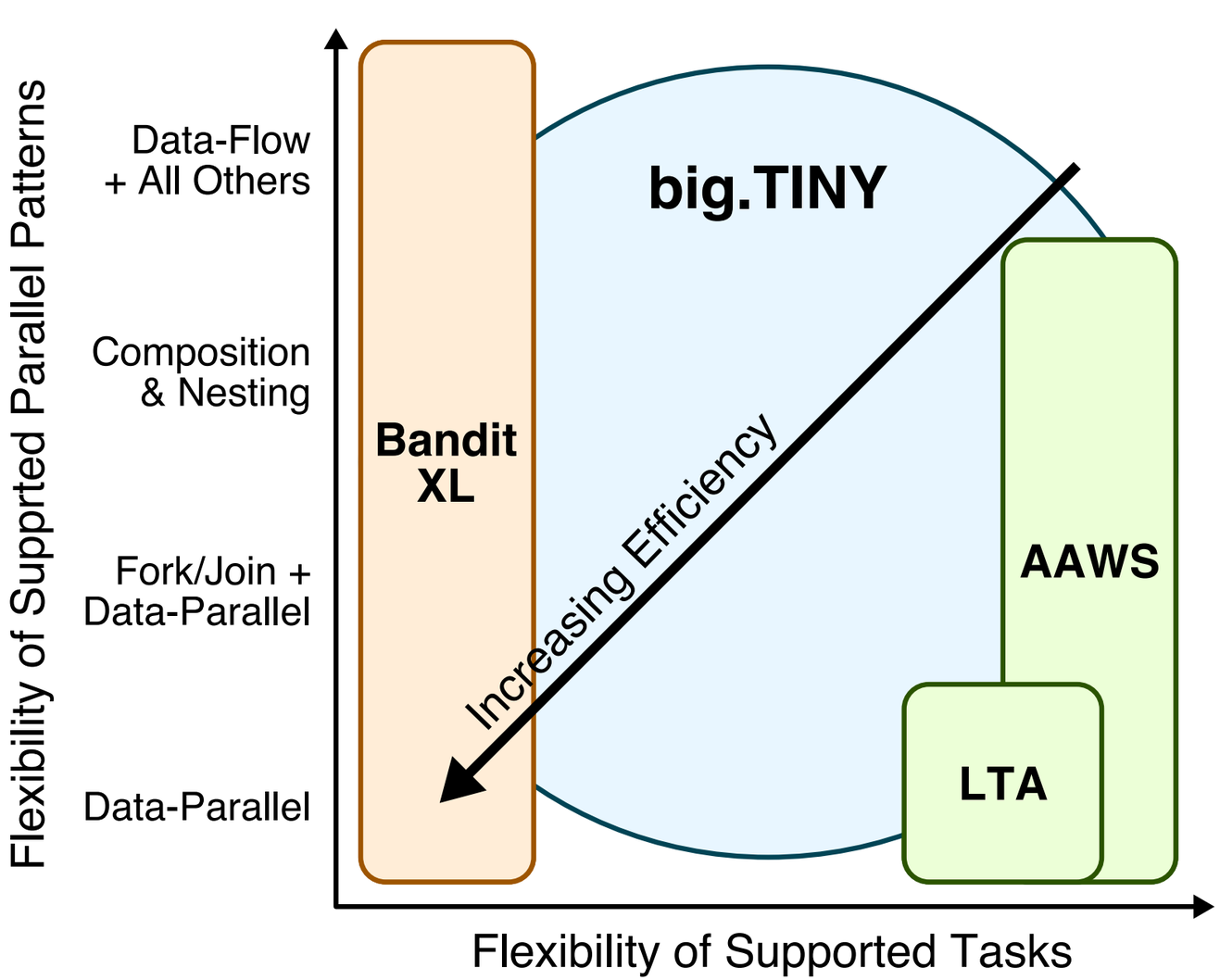


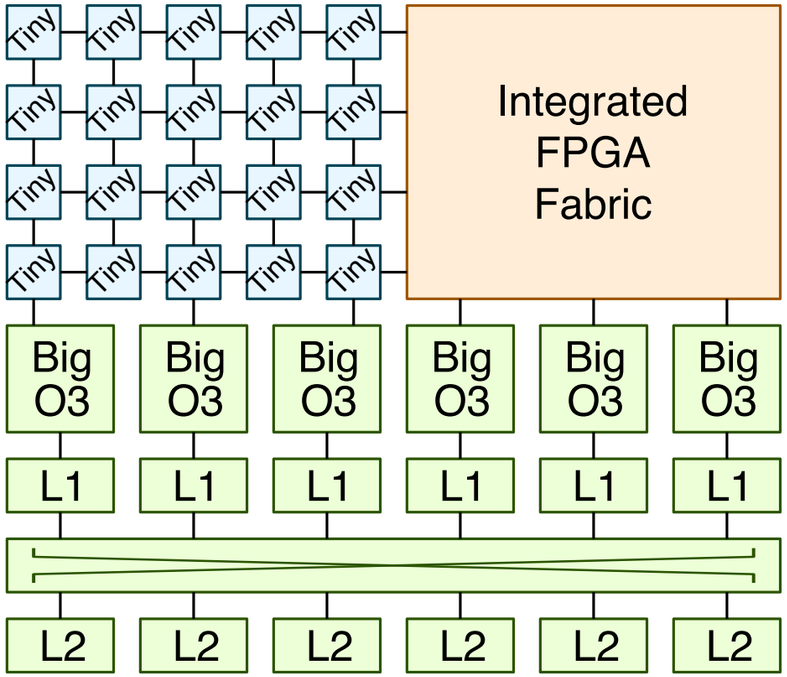
big.TINY: Balancing Flexibility and Efficiency



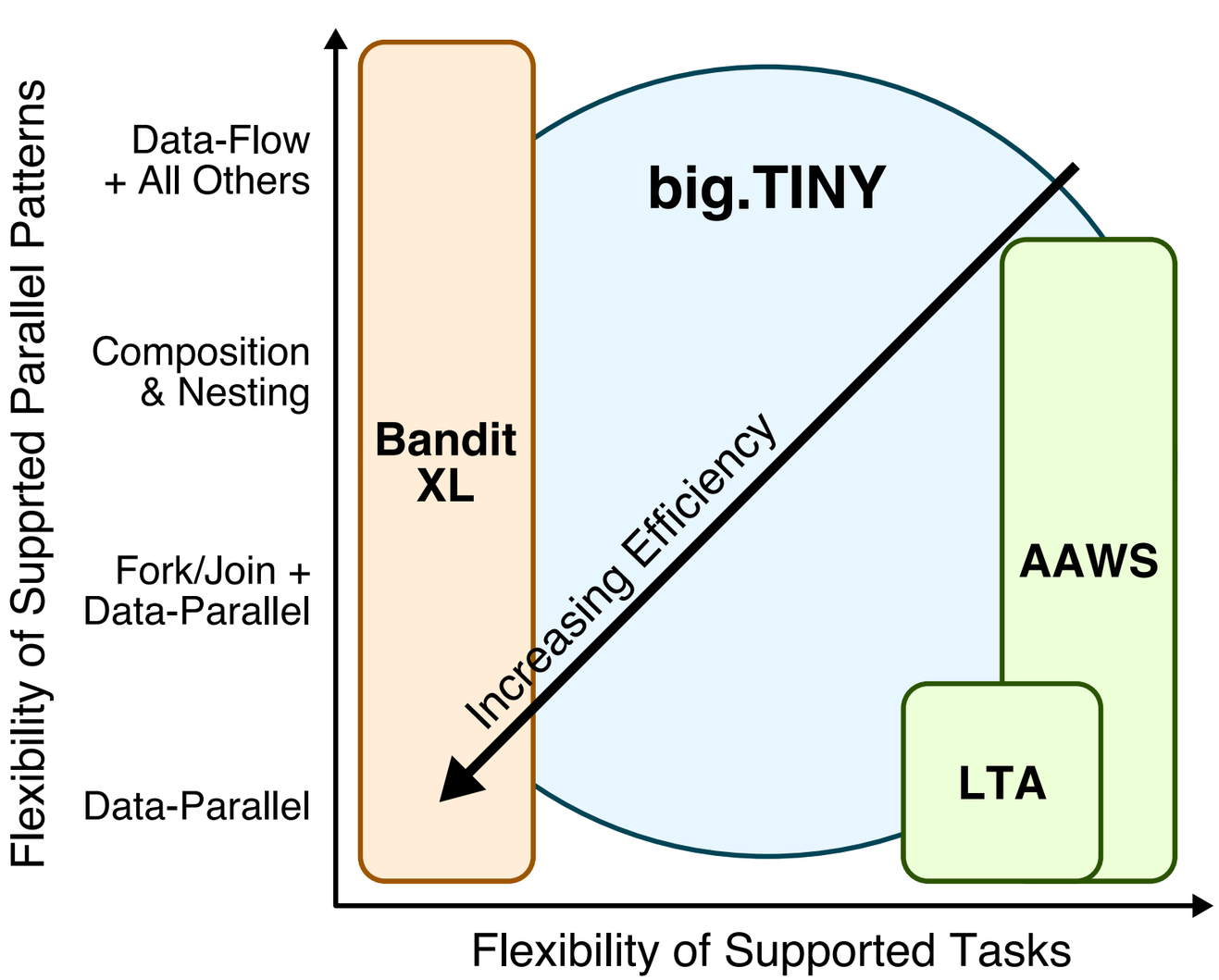


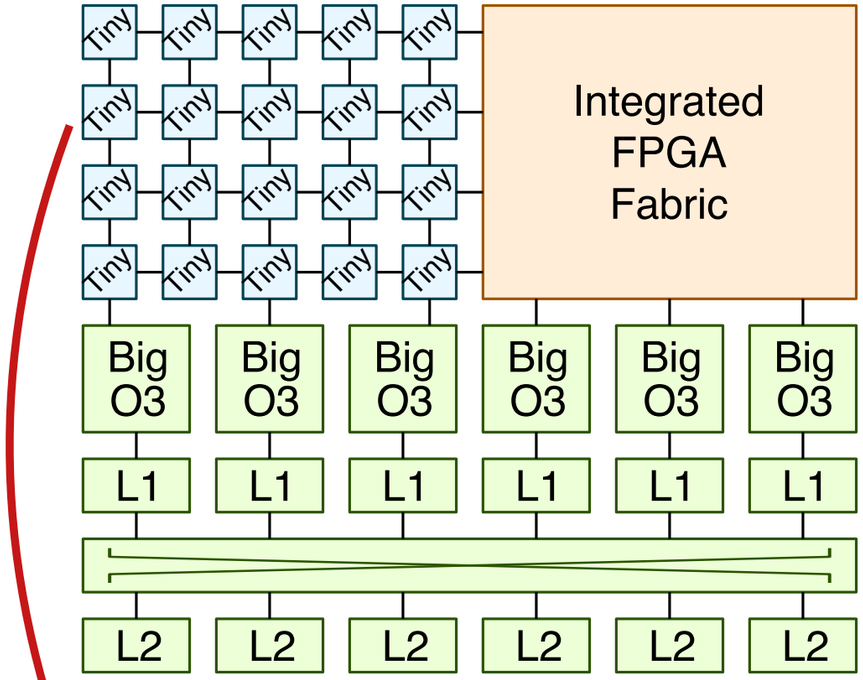
big.TINY: Balancing Flexibility and Efficiency

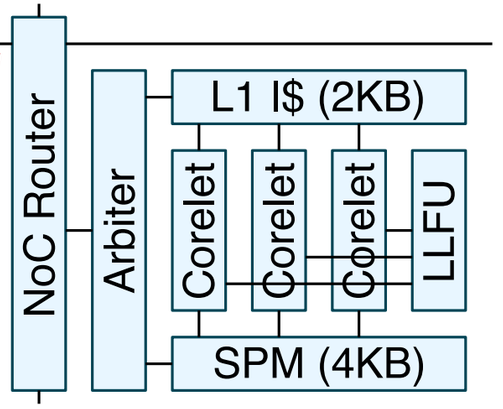




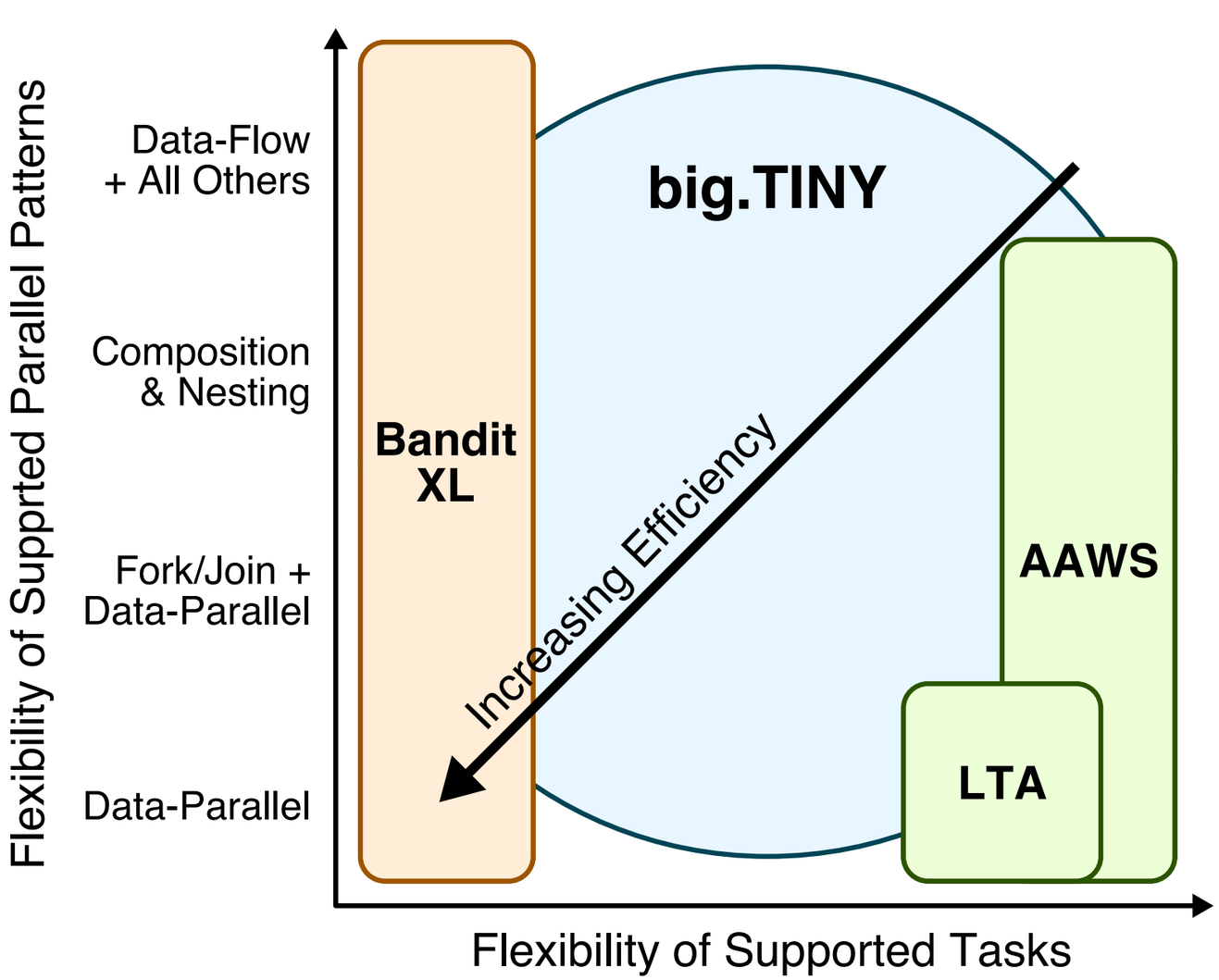
big.TINY: Balancing Flexibility and Efficiency

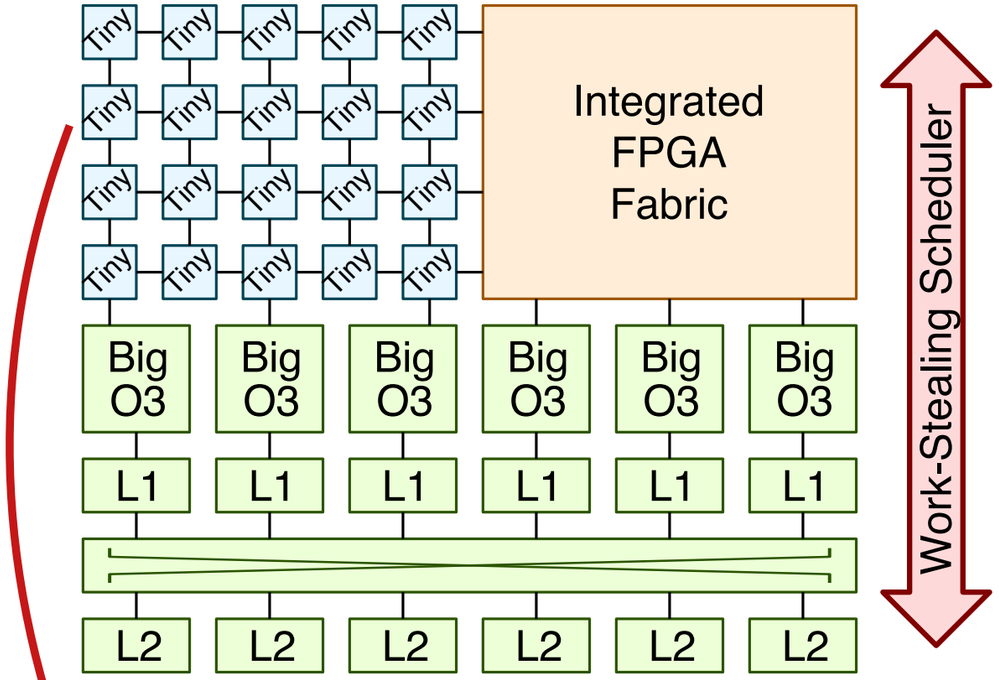


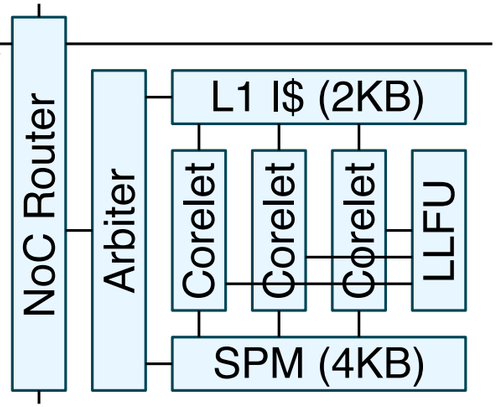




big.TINY: Balancing Flexibility and Efficiency

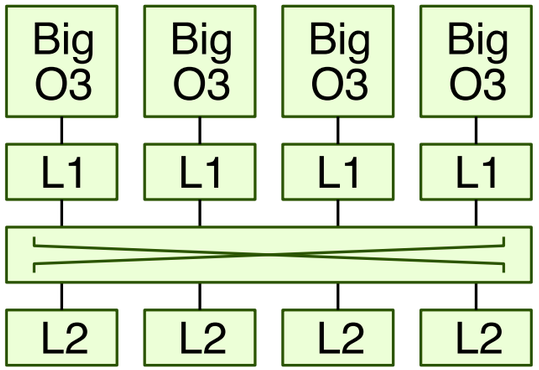


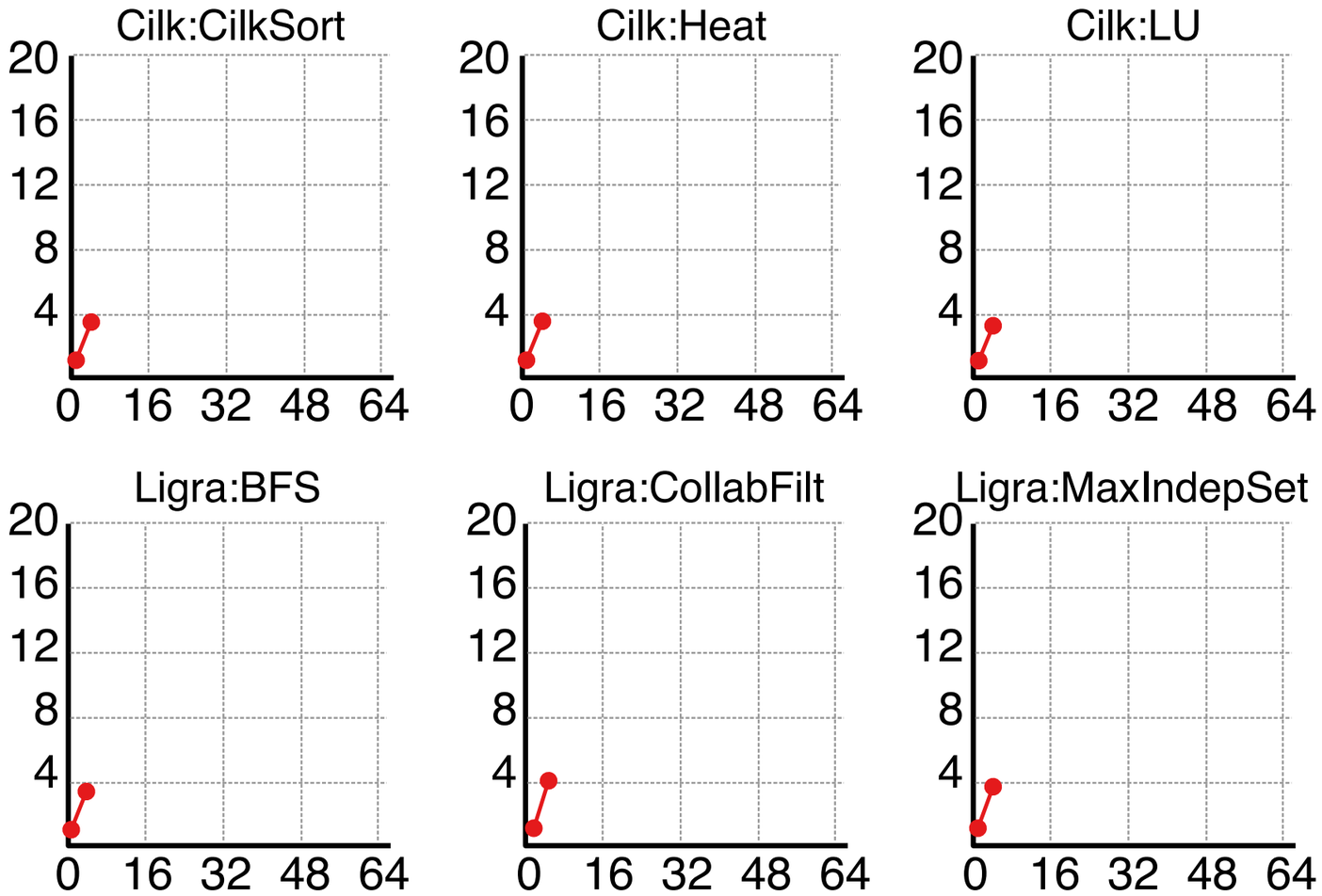




Enables common source or ISA abstraction across heterogenous SoC with a unified scheduling runtime

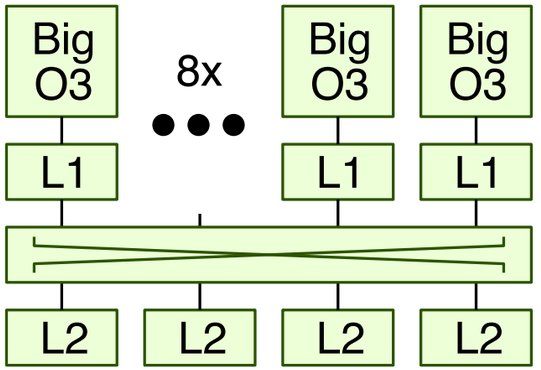
big.TINY: Balancing Flexibility and Efficiency

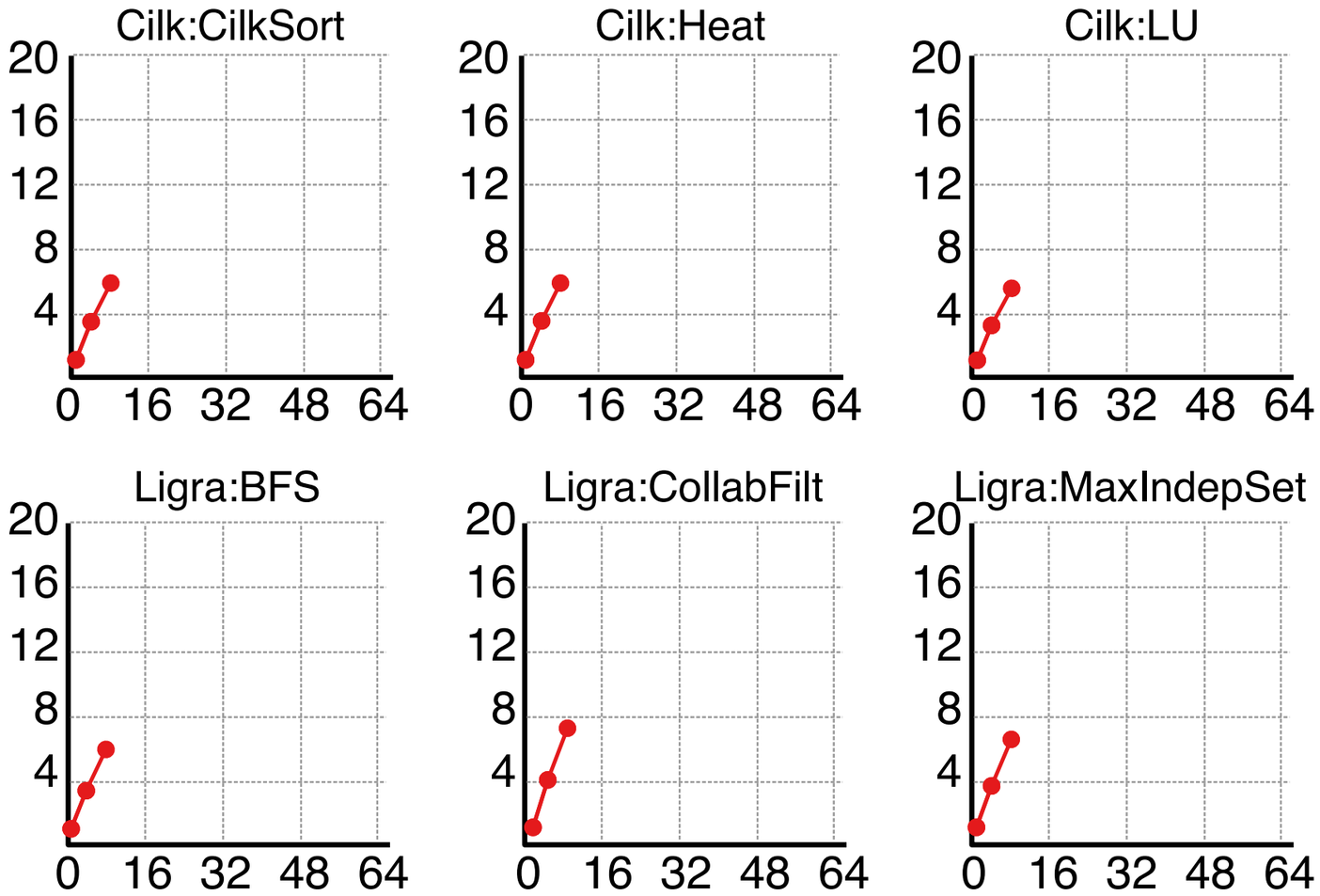




Applications Driving Architectures

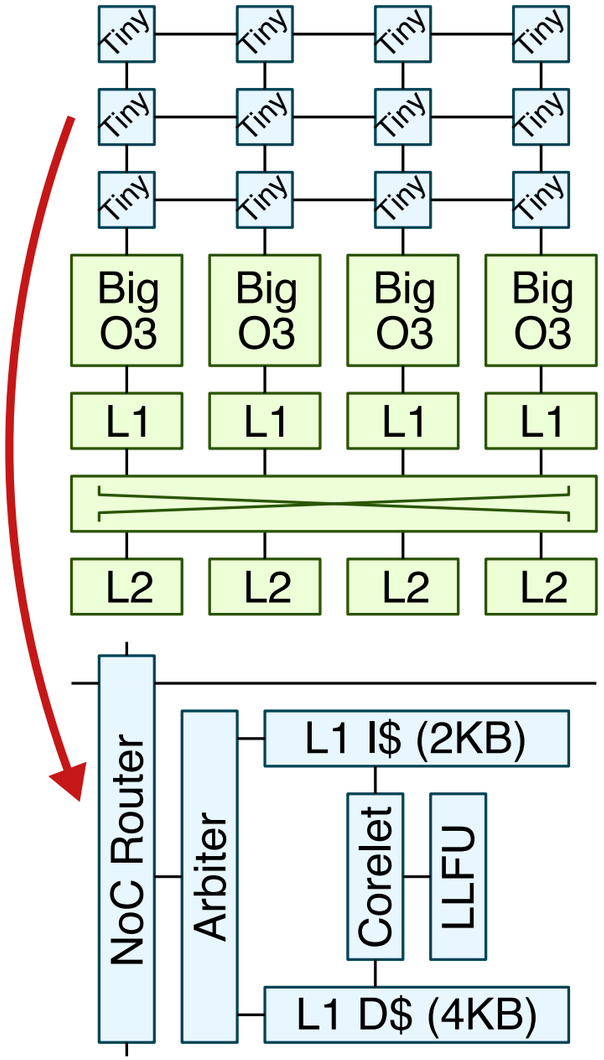
big.TINY: Balancing Flexibility and Efficiency

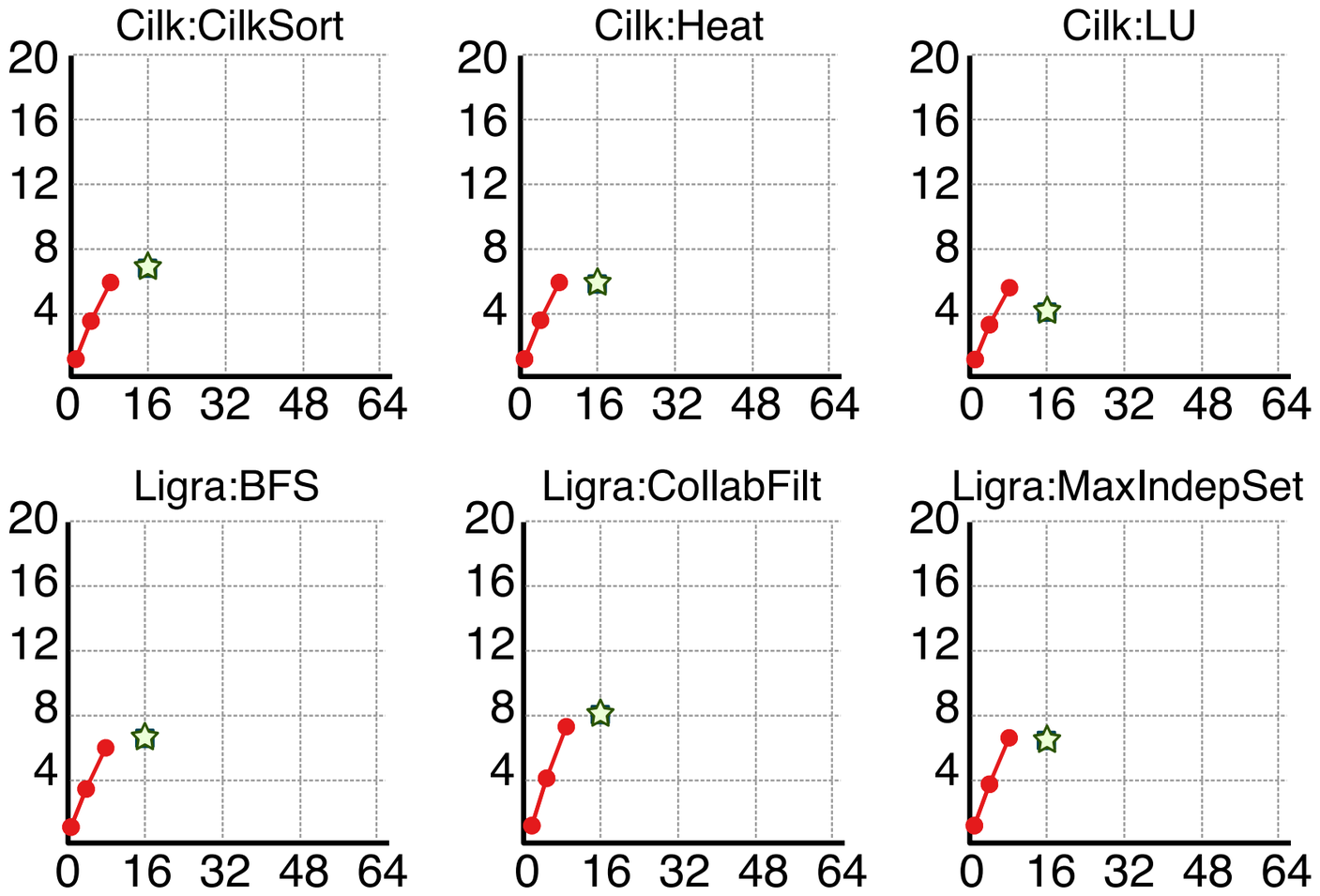




Applications Driving Architectures

big.TINY: Balancing Flexibility and Efficiency

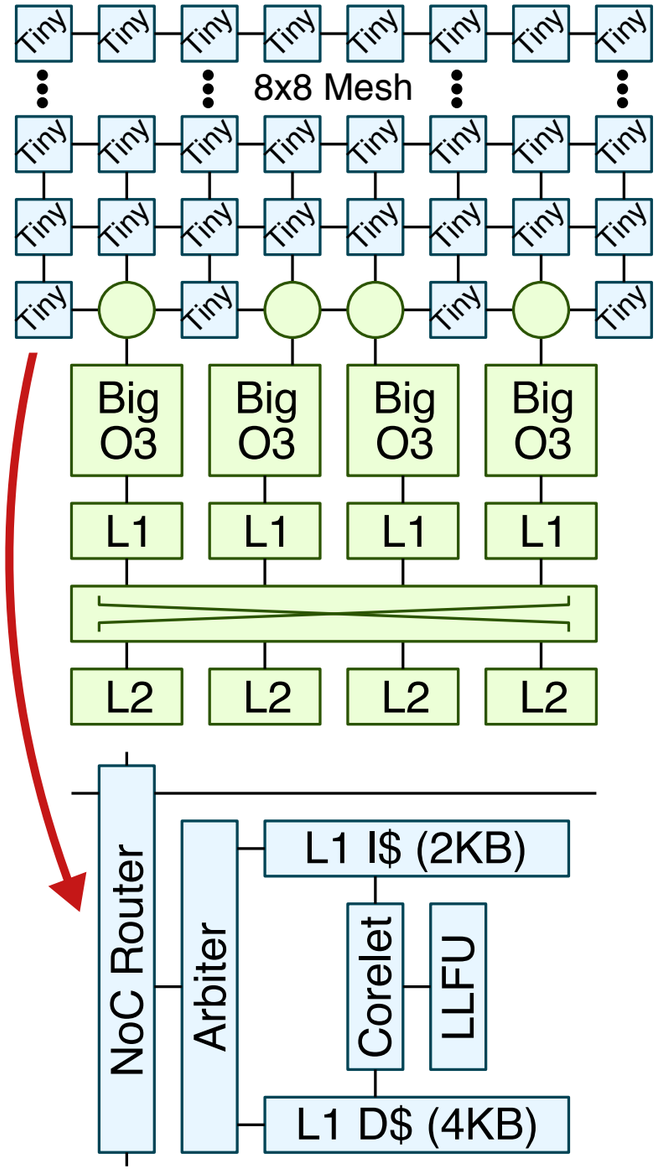


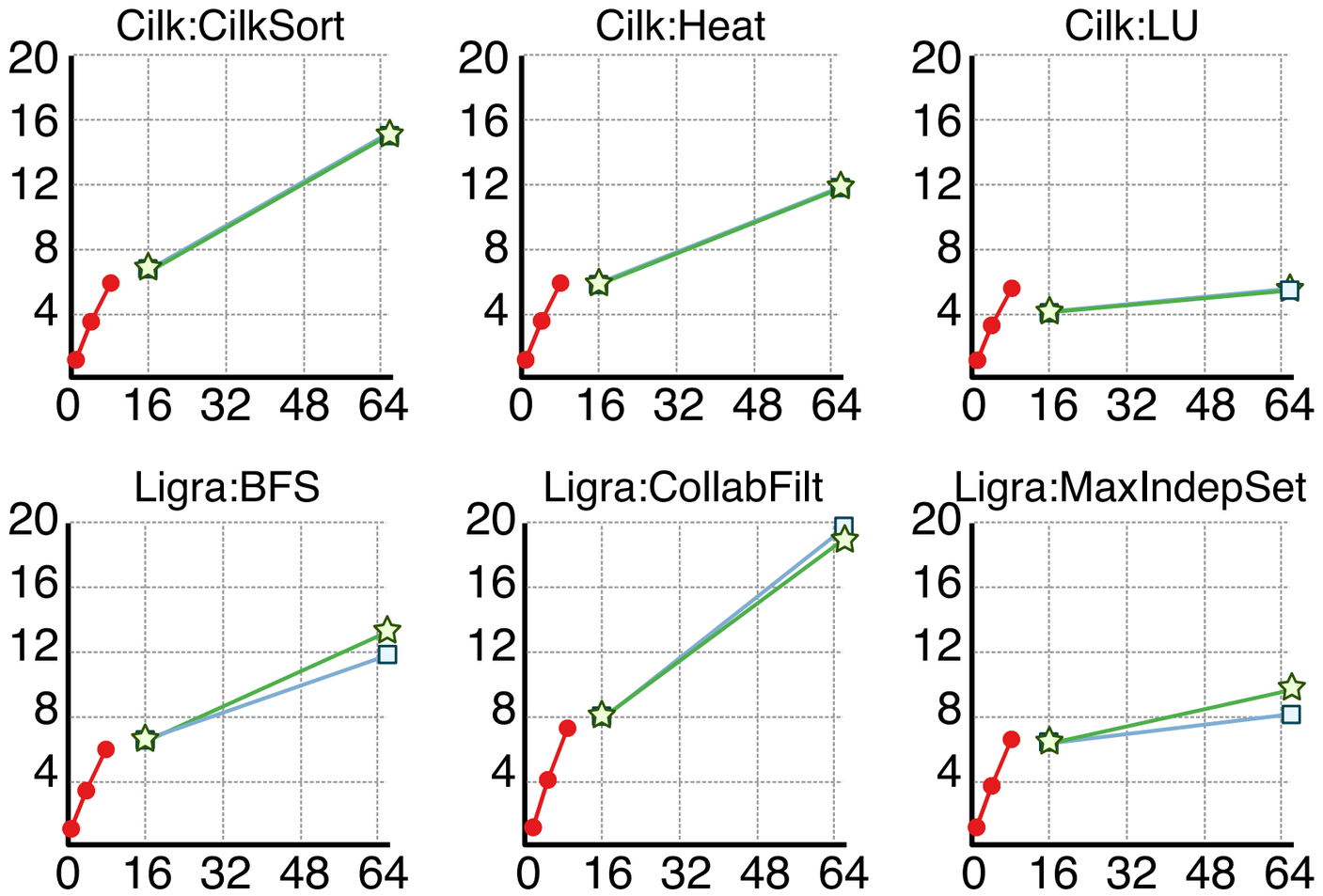


Tiny cores use small D$ with software-centric cache coherence

NOPe = no-ownership processing; sw flushes & invalidation; very, very simple

big.TINY: Balancing Flexibility and Efficiency





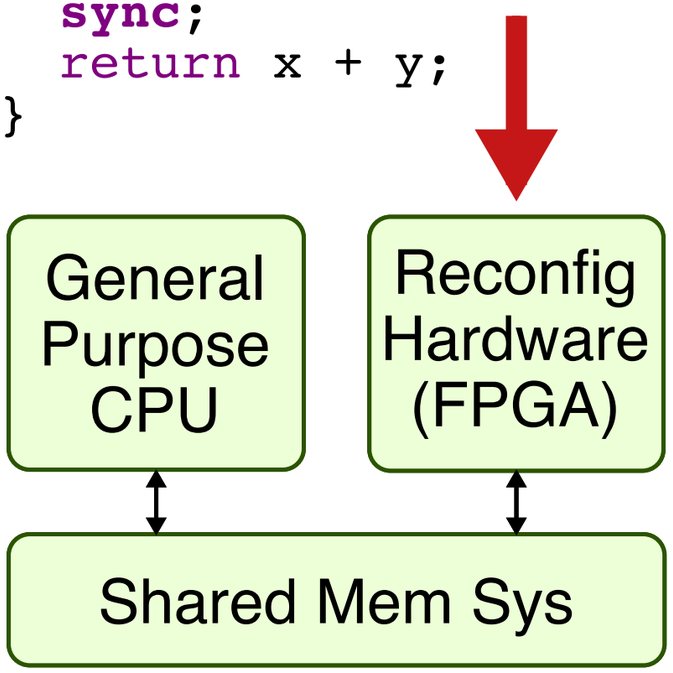
4 big cores require 384KB

60 tiny cores require 360KB

60 tiny cores have comparable area to adding four big cores

Architectural Specialization for Dynamic Parallel Algorithms and Work Stealing

```
int fib( int n )
{
  if (n < 2)
    return n;
  int x = spawn fib(n-1);
  int y = fib(n-2);
  sync;
  return x + y;
}
```



- Importance of exploring techniques for accelerating more complex applications on accelerator fabrics
- We have described a promising approach to accelerate dynamic parallel algorithms on FPGAs
  - computation model using explicit continuation passing
  - accelerator architecture based on work stealing
  - design methodology combining a PyMTL-based architectural template with high-level synthesis
- Future work in the center exploring big.TINY systems to optimally balance flexibility and efficiency
  - Connects to Spandex and GraphIT work within the center

JUMP

Joint University Microelectronics Program

www.src.org/program/jump

Semiconductor Research Corporation

@srcJUMP

Applications Driving Architectures