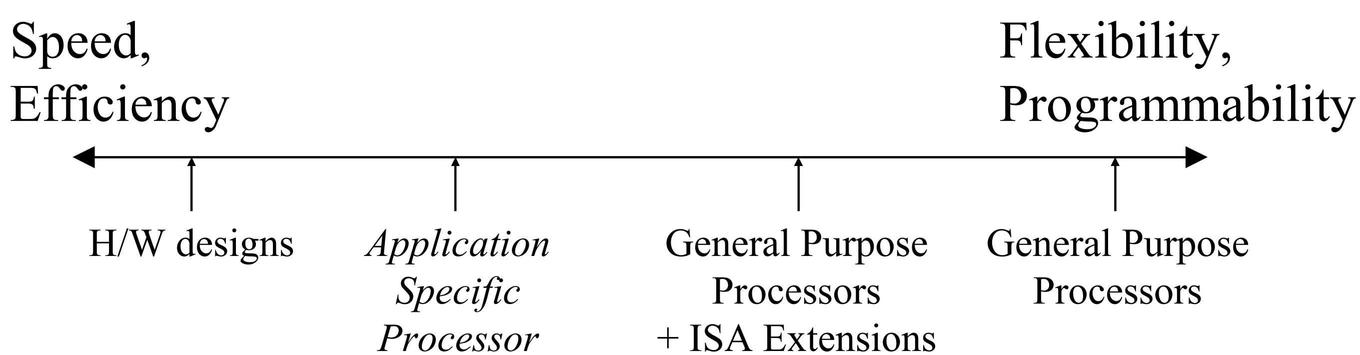
Summary: Benefits of App-Specific Design



- Specialization limits the scope of a device's operation
  - Produces stronger properties and invariants
  - Results in higher return optimizations
  - Programmability preserves the flexibility regarded by GPP's
- A natural fit for embedded designs
  - Where application domains are more likely restrictive
  - Where cost and power are 1st order concerns
- Overcomes growing silicon/architecture bottlenecks
  - Concentrated computation overcomes dark silicon dilemma
  - Customized acceleration speeds up Amdahl's serial codes

A Take on Composable Customization

Works presented here are from Jason Cong's research group @ UCLA

TCA vs LCA

- Tightly Coupled Accelerator (TCA)
  - Extended Instruction (e.g. MAC, SQRT)
  - Dedicated Accelerator (e.g. FFT, MPEG4)

- Loosely Coupled Accelerator (LCA)
  - Acts independently of individual cores
  - Can be shared among cores/resources
  - Essentially the “accelerator” we normally see.

LCA

- Dedicated: accelerator executes a program using domain-specific component.
  - Examples: GPU
- Programmable accelerator: Use programming fabrics to customized accelerator
  - Ex. FPGA-based accelerator
- Composable: combines accelerator building blocks into an accelerator

Why bother composing accelerator?

- Dark silicon provide extra area for incorporating more accelerator?
  - Yes… but how many accelerator do we really need?
  - An LCA may be useless for new algorithms or new domains
  - Essentially, it is not practical to build an accelerator for every single application
- LCA is
  - Often under-utilized
  - Contain many replicated structures (things like fp-ALUs, DMA engines, SPM)
    - Unused when the accelerator is unused

How do we compose an accelerator?

- ABB (Accelerator building block)
  - A Block of accelerator unit that performs small specific task

ABBs	Denoise	Deblur	Registration	Segmentation
Float Reciprocal (FInv)	✓	✓		✓
Float Square-Root (FSqrt)	✓	✓	✓	✓
Float Polynomial-16 (Poly16)	✓	✓	✓	✓
Float Divide (FDiv)	✓	✓	✓	✓

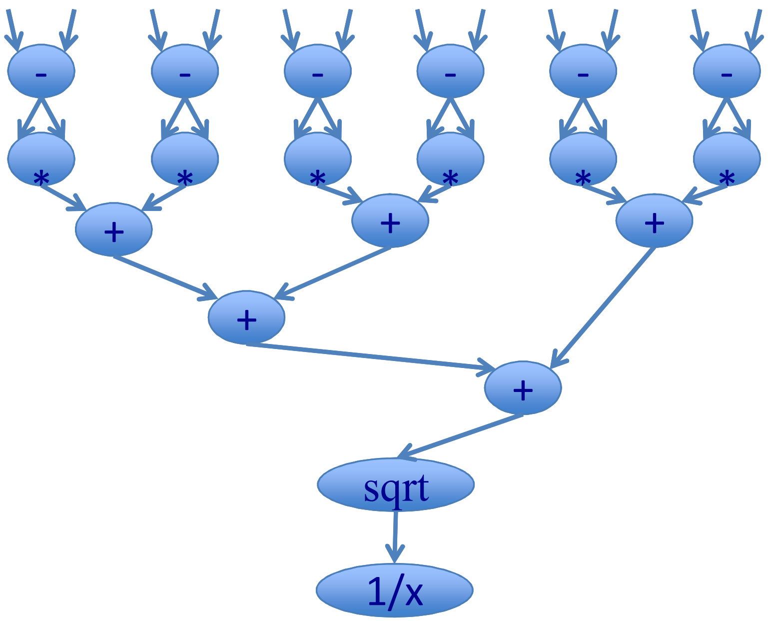
From CHARM: A Composable Heterogeneous Accelerator-Rich Microprocessor ISLPED'12

Example of ABB Flow-Graph (Denoise)

$$\mathbf{1}\Big/\sqrt{\sum_{i=0}^{6}(\boldsymbol{Xi}-\boldsymbol{Y})^{2}}$$

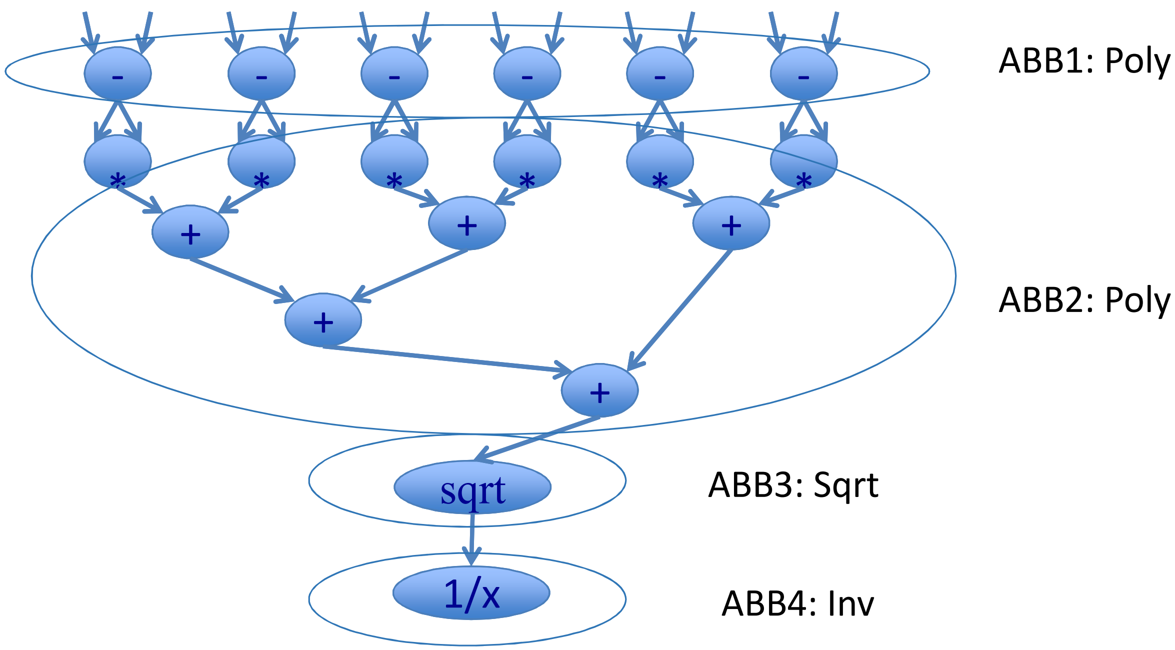
Example of ABB Flow-Graph (Denoise)

$$1 \Big/ \sqrt{\sum_{i=0}^{6} (Xi - Y)^2}$$



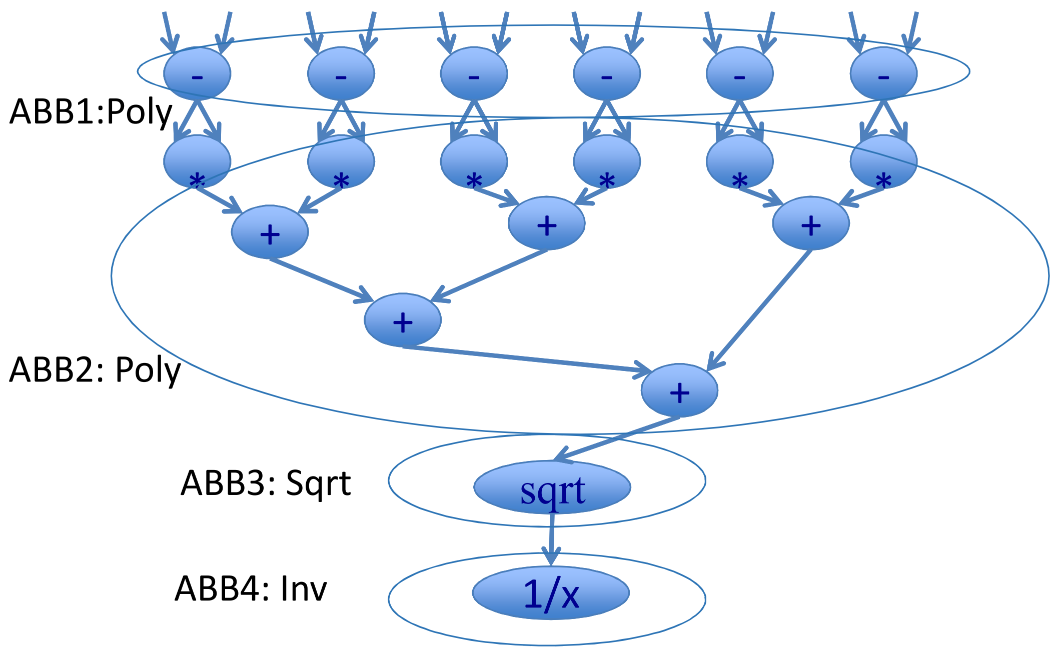
Example of ABB Flow-Graph (Denoise)

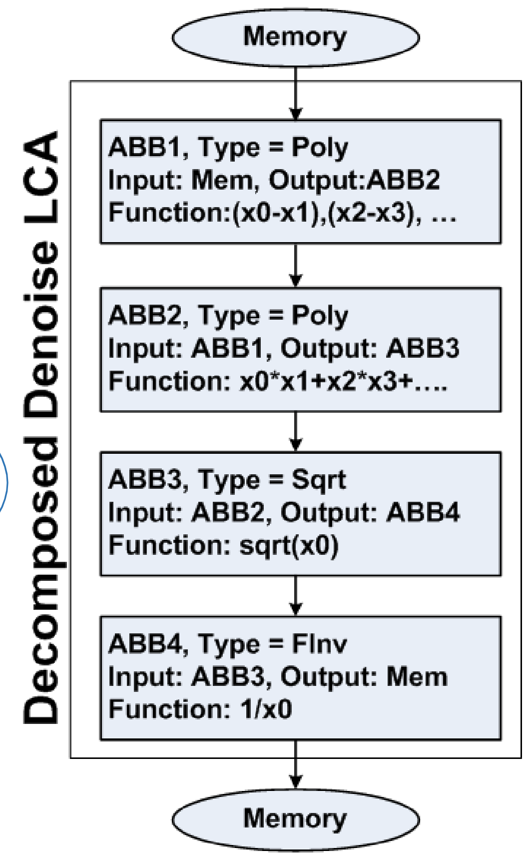
$$1/\sqrt{\sum_{i=0}^{6}(Xi-Y)^2}$$



Example of ABB Flow-Graph (Denoise)

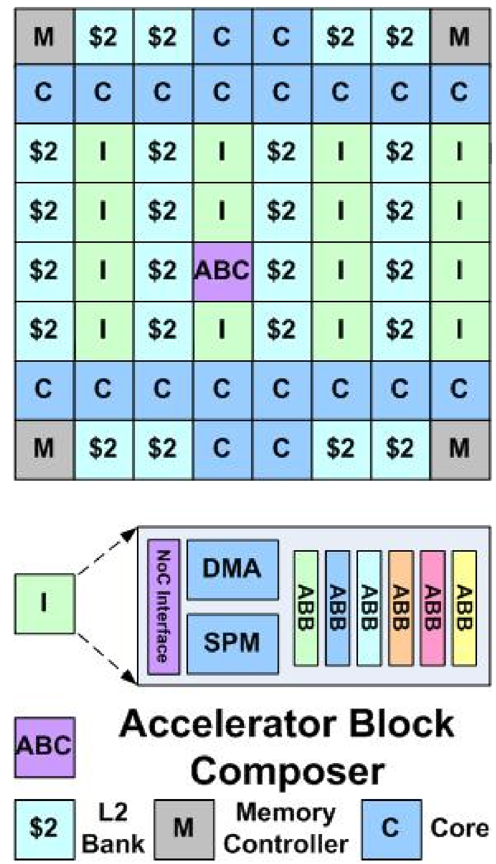
$$1 / \sqrt{\sum_{i=0}^{6} (Xi - Y)^2}$$





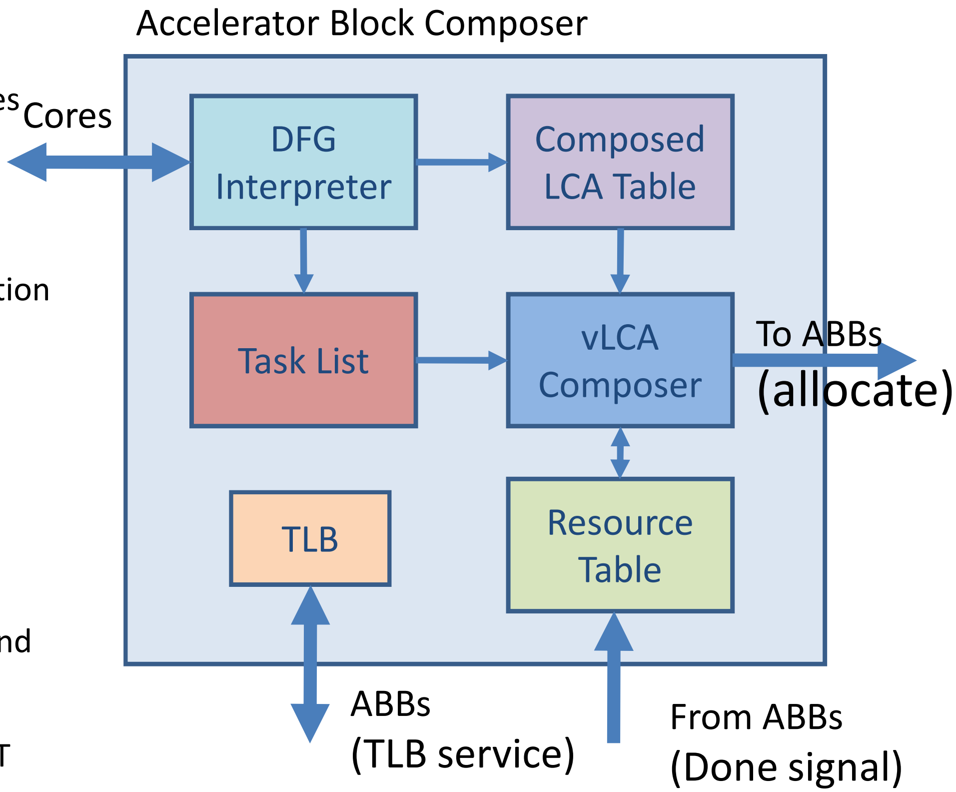
Micro Architecture of CHARM

- ABB
  - **Accelerator Building Blocks (ABB)**
  - **Primitive components that can be composed into accelerators**
- ABB islands
  - **Multiple ABBs**
  - **Shared DMA controller, SPM and NoC interface**
- ABC
  - **Accelerator Block Composer (ABC)**
    - **To orchestrate the data flow between ABBs to create virtual accelerator**
    - **Arbitrate requests from cores**
- Other components
  - **Cores**
  - **L2 Banks**
  - **Memory controllers**



ABC Internal Design

- ABC sub-components
  - **Resource Table(RT)**: To keep track of available/used ABBs
  - **Composed LCA Table (CLT):** Eliminates the need to re-compose virtual LCAs
  - **Task List (TL):** To queue the broken virtual LCA requests (to smaller data size)
  - **TLB**: To service and share the translation requests by ABBs
  - **Task Flow-Graph Interpreter (TFGI):** Breaks the virtual LCA DFG into ABBs
  - **vLCA Composer (vLC):** Compose the virtual LCA using available ABBs
- Implementation
  - **RT, CLT, TL and TLB** are implemented using **RAM**
  - **TFGI** has a **table** to keep ABB types and an **FSM** to read task-flow-graph and compares
  - **vLC** has an **FSM** to go over CLT and RT and check mark the available ABBs

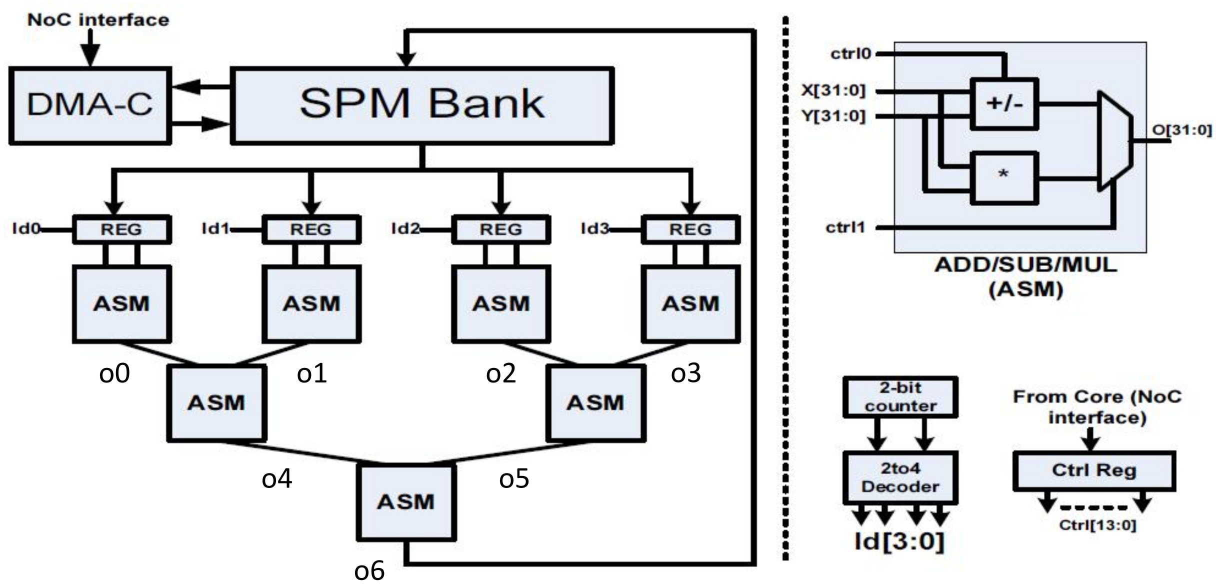


An Example of ABB Library (for Medical Imaging)

ABBs	Denoise	Deblur	Registration	Segmentation
Float Reciprocal (FInv)	✓	✓		✓
Float Square-Root (FSqrt)	✓	✓	✓	✓
Float Polynomial-16 (Poly16)	✓	✓	✓	✓
Float Divide (FDiv)	✓	✓	✓	✓

Internal

of Poly



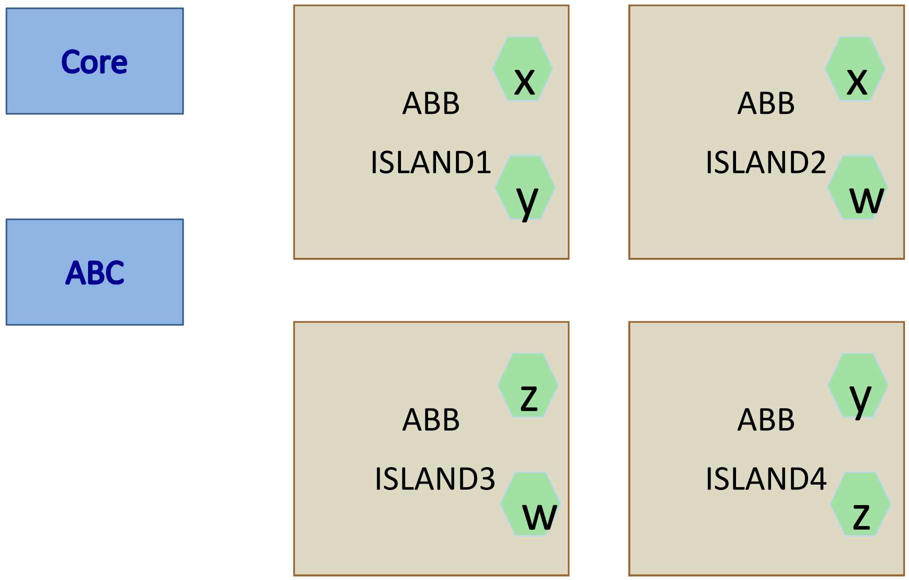
Virtual LCA Composition Process

All islands have X, Y, Z, W

For Simplicity only those

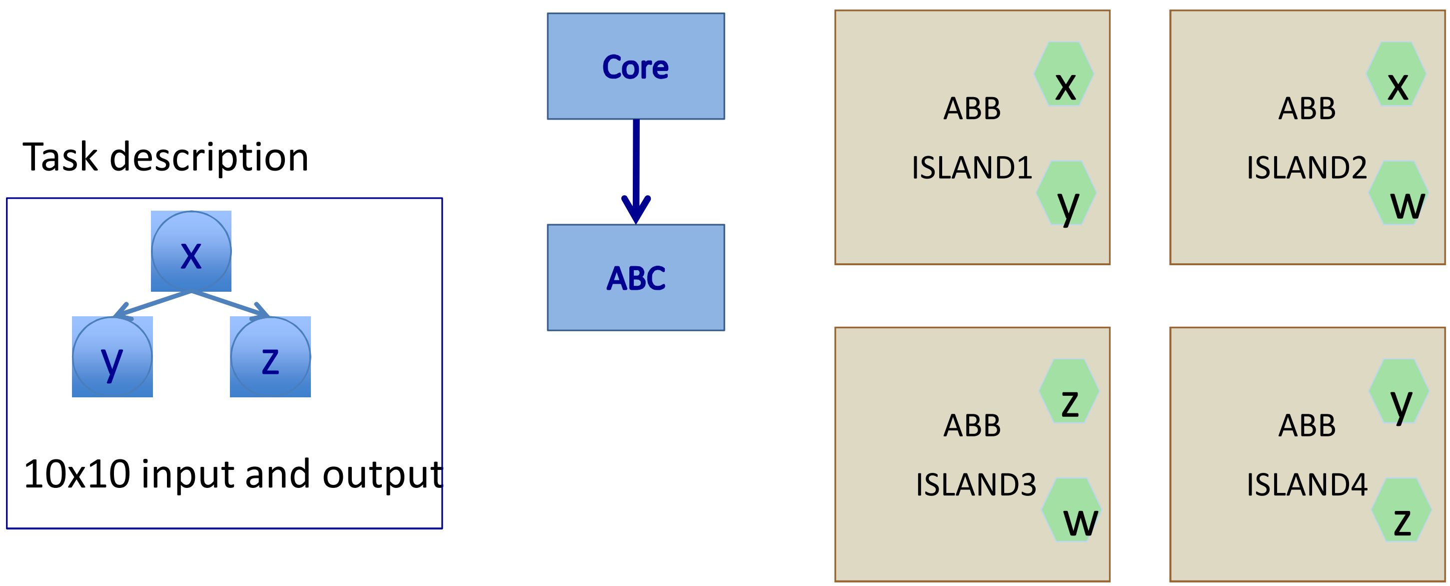
ABBs which are available

now are shown



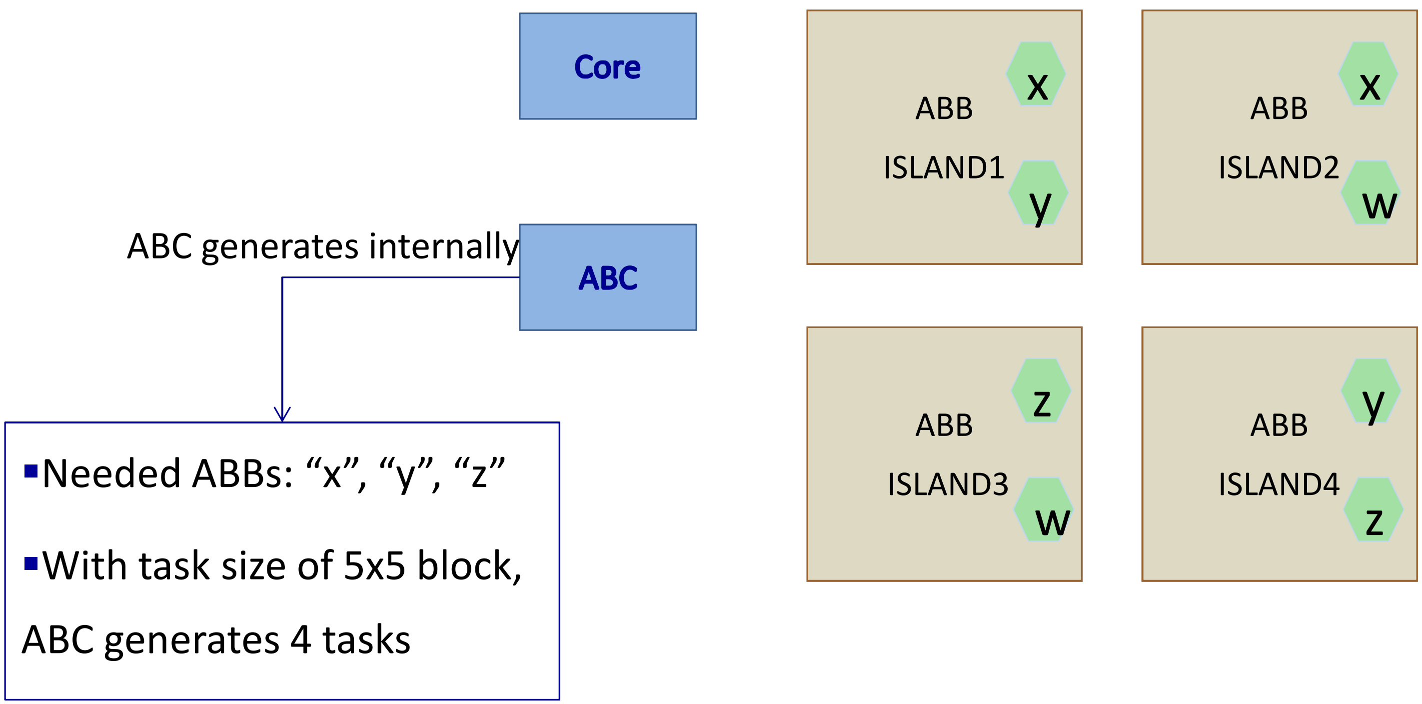
Virtual LCA Composition Process

1. Core initiation
   - Core sends the task description: task flow-graph of the desired LCA to ABC together with polyhedral space for input and output



Virtual LCA Composition Process

2. Task-flow parsing and task-list creation
   - ABC parses the task-flow graph and breaks the request into a set of tasks with smaller data size and fills the task list

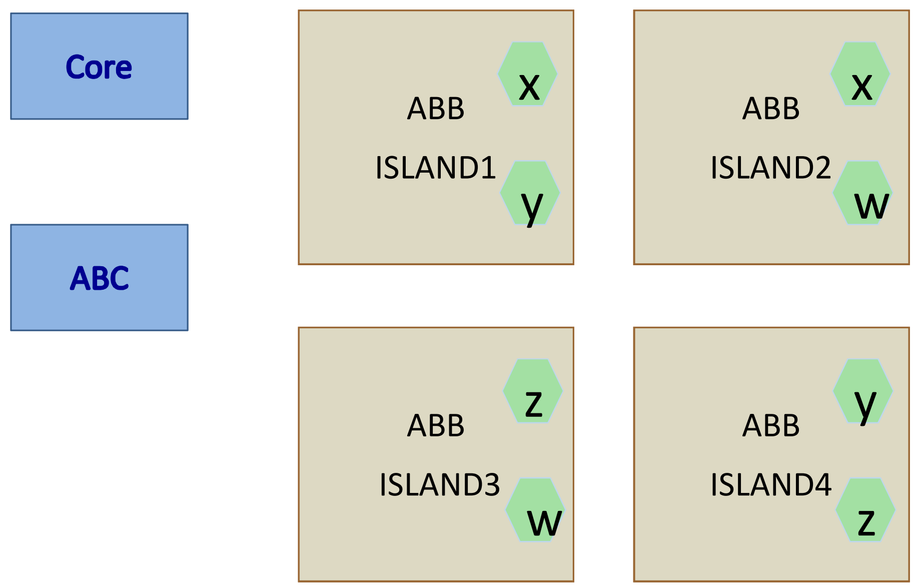


Virtual LCA Composition Process

3. Dynamic ABB mapping

- ABC uses a pattern matching algorithm to assign ABBs to islands
- Fills the composed LCA table and resource allocation table

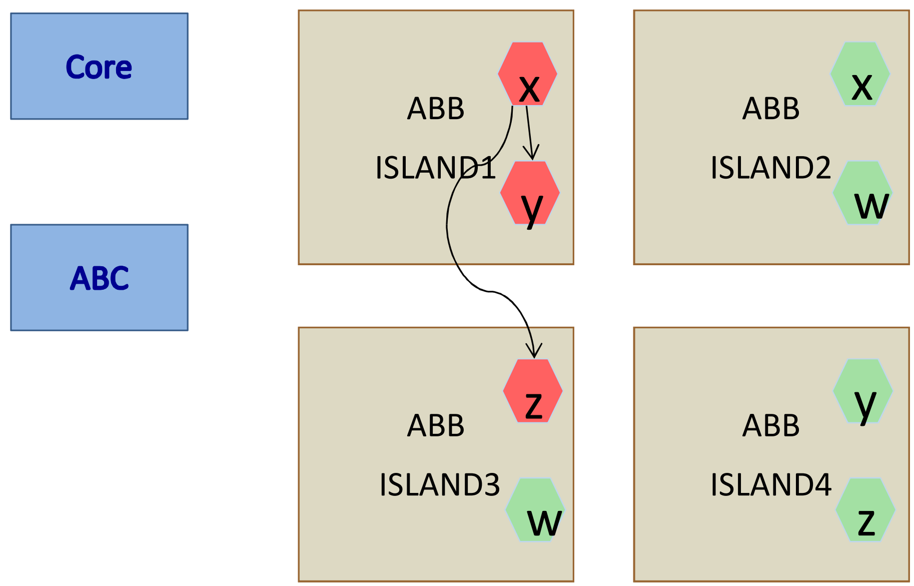
Island ID	ABB Type	ABB ID	Status
1	x	1	Free
1	y	1	Free
2	x	1	Free
2	w	1	Free
3	z	1	Free
3	w	1	Free
4	y	1	Free
4	z	1	Free



Virtual LCA Composition Process

3. Dynamic ABB mapping
   - ABC uses a pattern matching algorithm to assign ABBs to islands
   - Fills the composed virtual LCA table and resource allocation table

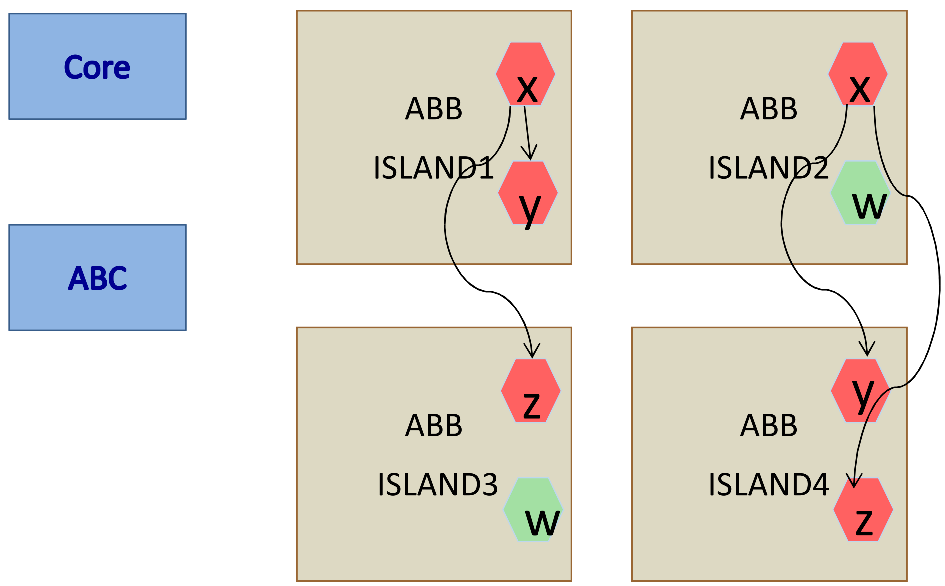
Island ID	ABB Type	ABB ID	Status
1	x	1	Busy
1	y	1	Busy
2	x	1	Free
2	w	1	Free
3	z	1	Busy
3	w	1	Free
4	y	1	Free
4	z	1	Free



Virtual LCA Composition Process

4. LCA cloning
    - Repeat to generate more virtual LCAs if ABBs are available

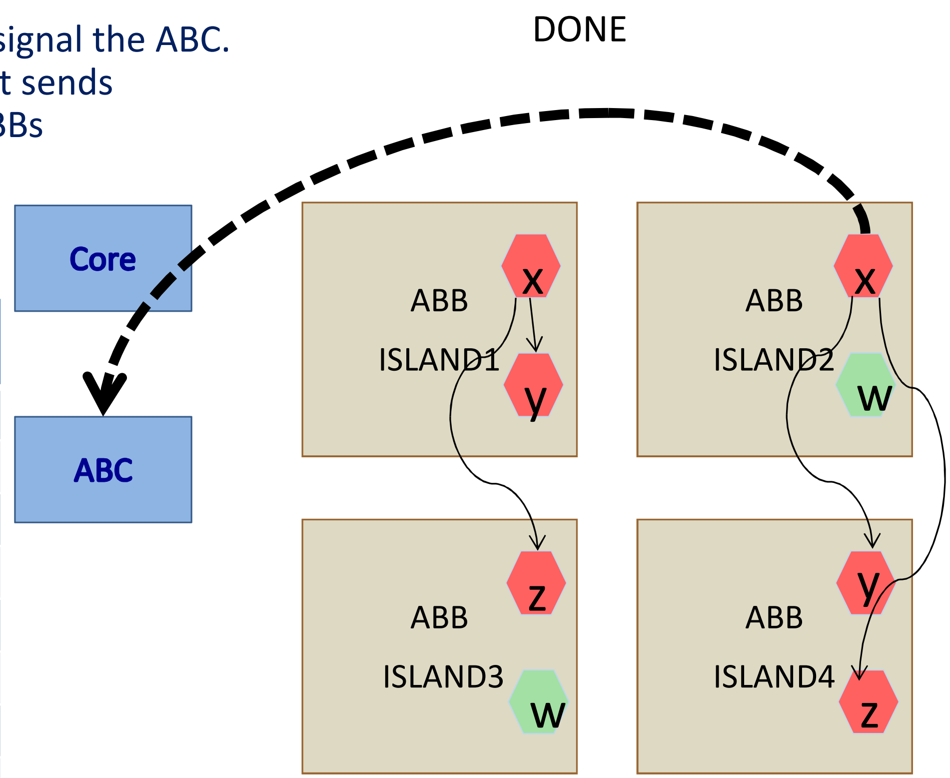
Core ID	ABB Type	ABB ID	Status
1	x	1	Busy
1	y	1	Busy
2	x	1	Busy
2	w	1	Free
3	z	1	Busy
3	w	1	Free
4	y	1	Busy
4	z	1	Busy



Virtual LCA Composition Process

5. ABBs finishing task
   - When ABBs finish, they signal the ABC. If ABC has another task it sends otherwise it frees the ABBs

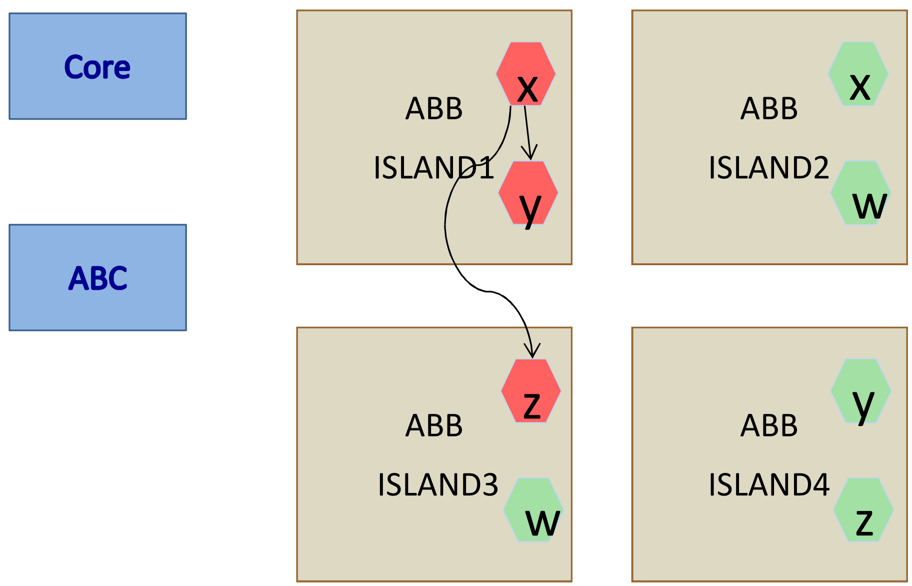
Island ID	ABB Type	ABB ID	Status
1	x	1	Busy
1	y	1	Busy
2	x	1	Busy
2	w	1	Free
3	z	1	Busy
3	w	1	Free
4	y	1	Busy
4	z	1	Busy



Virtual LCA Composition Process

5. ABBs being freed
   - When an ABB finishes, it signals the ABC. If ABC has another task it sends otherwise it frees the ABBs

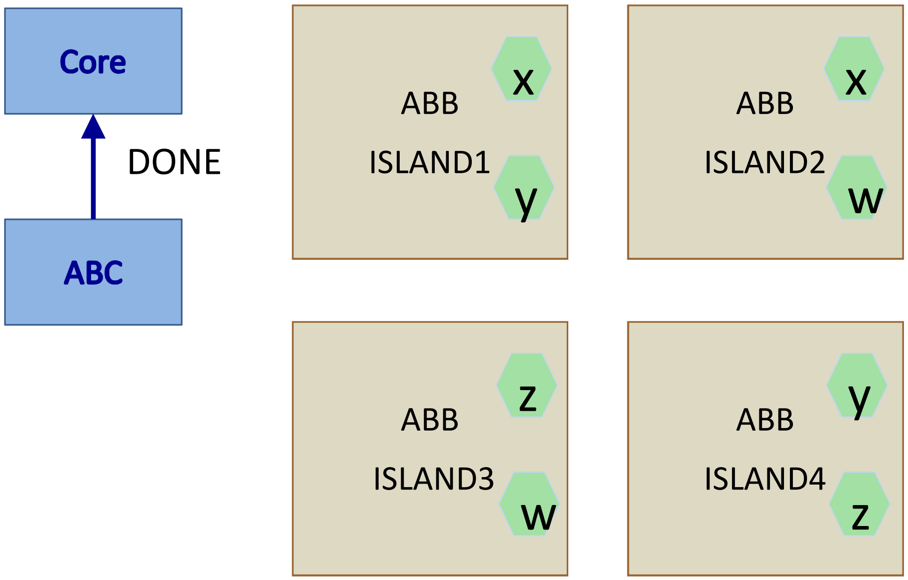
Island ID	ABB Type	ABB ID	Status
1	x	1	Busy
1	y	1	Busy
2	x	1	Free
2	w	1	Free
3	z	1	Busy
3	w	1	Free
4	y	1	Free
4	z	1	Free



Virtual LCA Composition Process

6. Core notified of end of task
   - When the virtual LCA finishes ABC signals the core

Island ID	ABB Type	ABB ID	Status
1	x	1	Free
1	y	1	Free
2	x	1	Free
2	w	1	Free
3	z	1	Free
3	w	1	Free
4	y	1	Free
4	z	1	Free



Limitation?

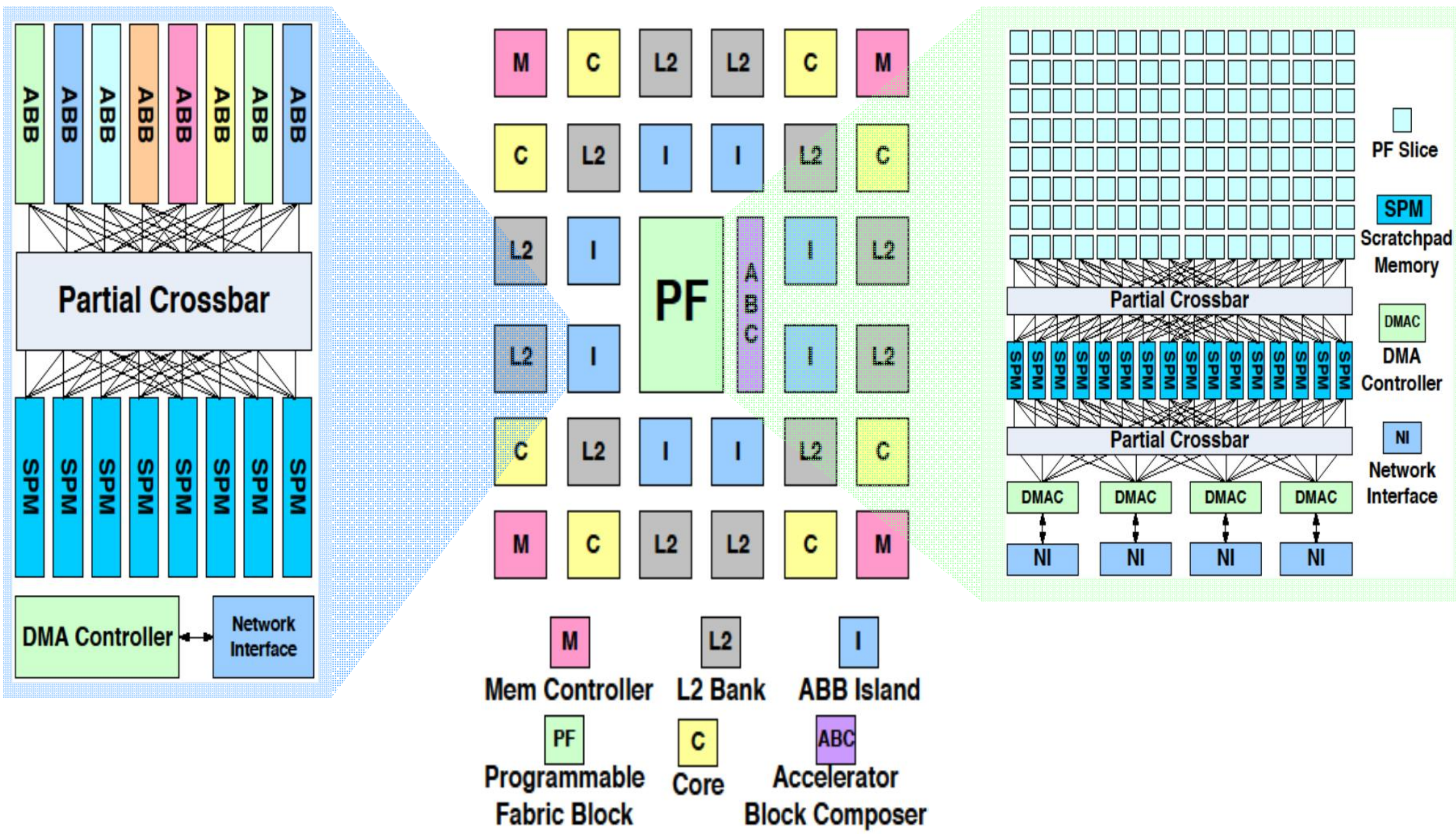
- Composing accelerator from building blocks still only serve limited range of applications
  - So incorporate Programmable fabric

ASICS vs. Programmable Accelerator

+ Fast
+ Small Area (per accelerator)
+ Energy Efficient
- Inflexible
- Need more as applications become diverse

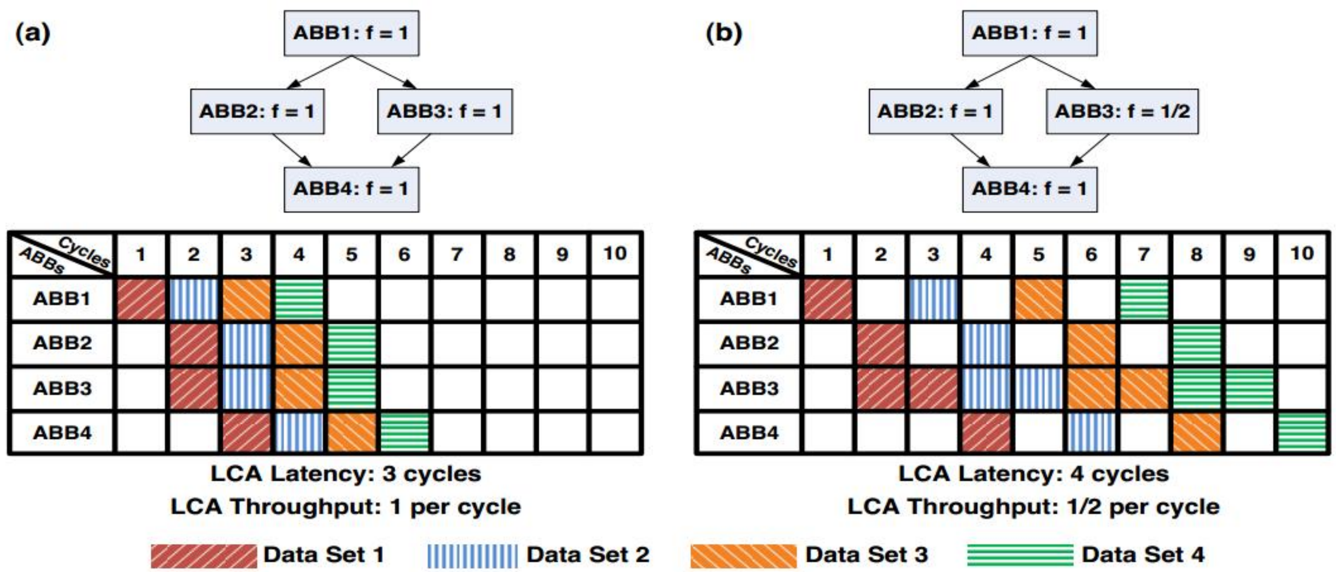
Pretty much the opposite
+ Reconfigurable
+ Small Area (Overall)
+ Good Utilization
- Not Efficient
- Slower than ASICs

CAMEL Architecture (ISLPED'13)



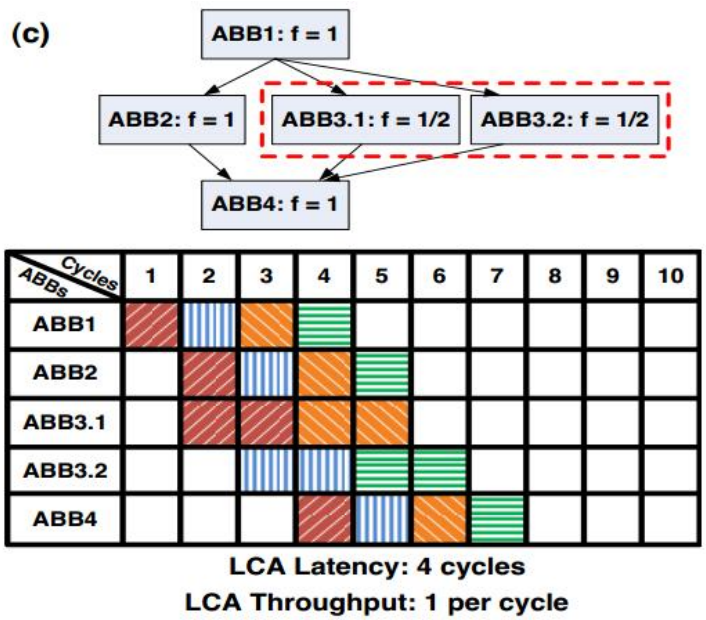
Challenges in incorporating PF

- Operating accelerators with different speeds (frequencies) can create a bottleneck. Especially, since PFs are slower than ABBs.



Rate-Matching Technique

- Duplicates slow accelerators to bring up throughput



Runtime PF Allocations

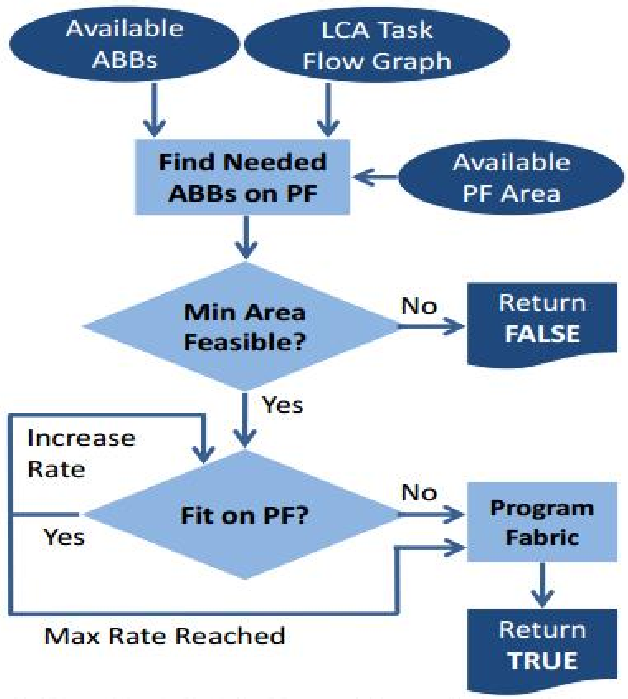


Fig. 5: PF Allocation Algorithm

Compiler Framework

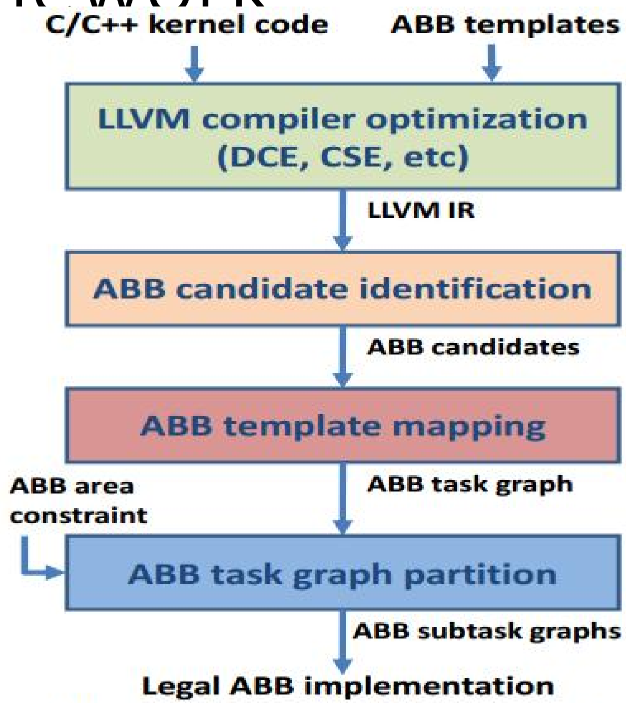


Fig. 6: Compiler Framework

Note that is kernel being mapped is too large for total # of ASICs + PFs, task flow graph is partitioned (in a way that minimize data transfer)

Result?

- 11.6X performance improvement, 13.9X energy savings over CHARM (up to over 30X from GP)
- Experimental results found optimal percentage of PFs to be around 30% for application domain like Medical imaging/Navigation, and 50% for commercial application domain and computer visions
- Still more work to be done!

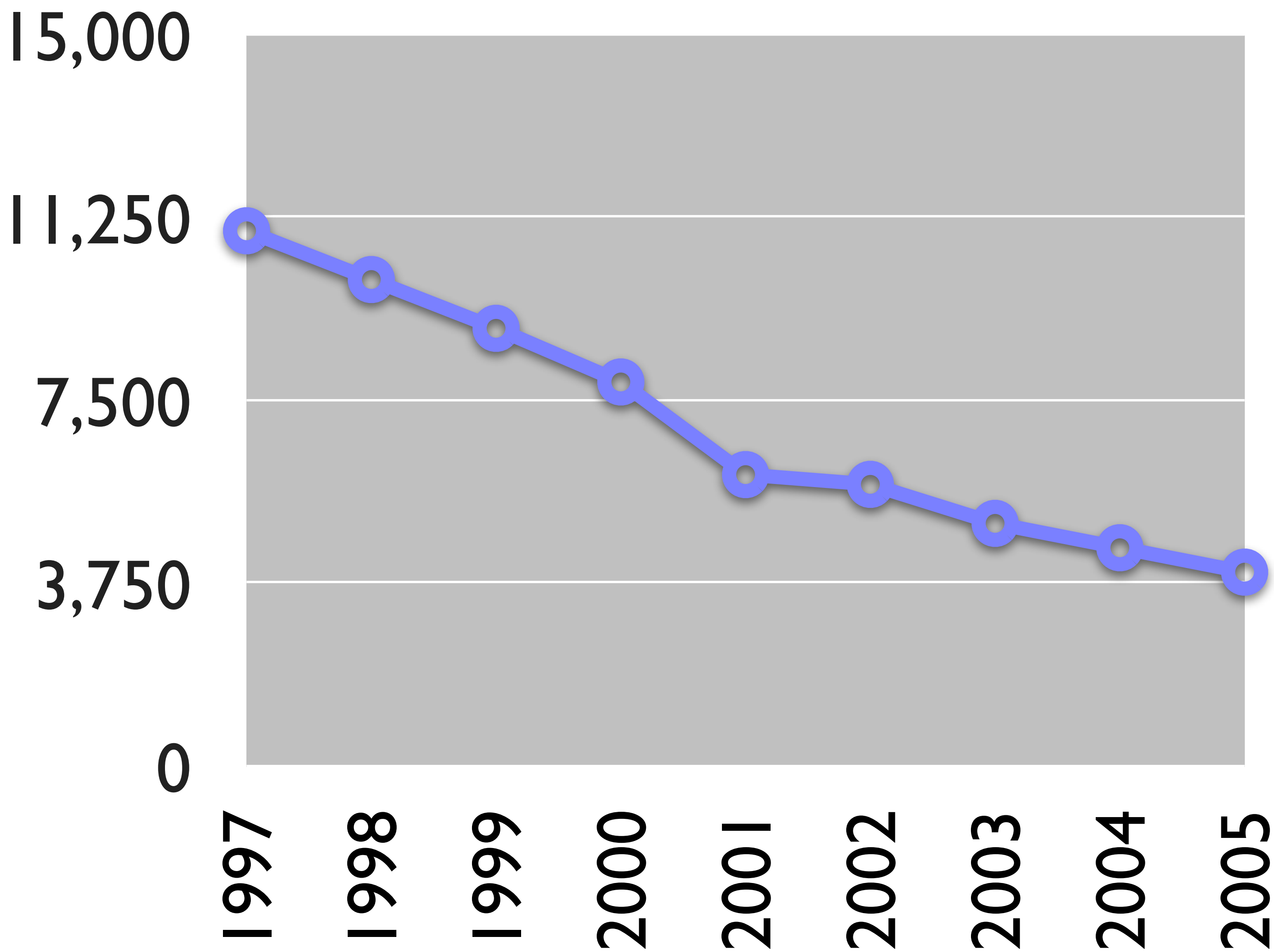
Brick and Mortar Silicon Manufacturing

Martha Mercaldi
Mark Oskin, Todd Austin, Karl Bohringer, Azita Emami

University of Washington, University of Michigan, Columbia University

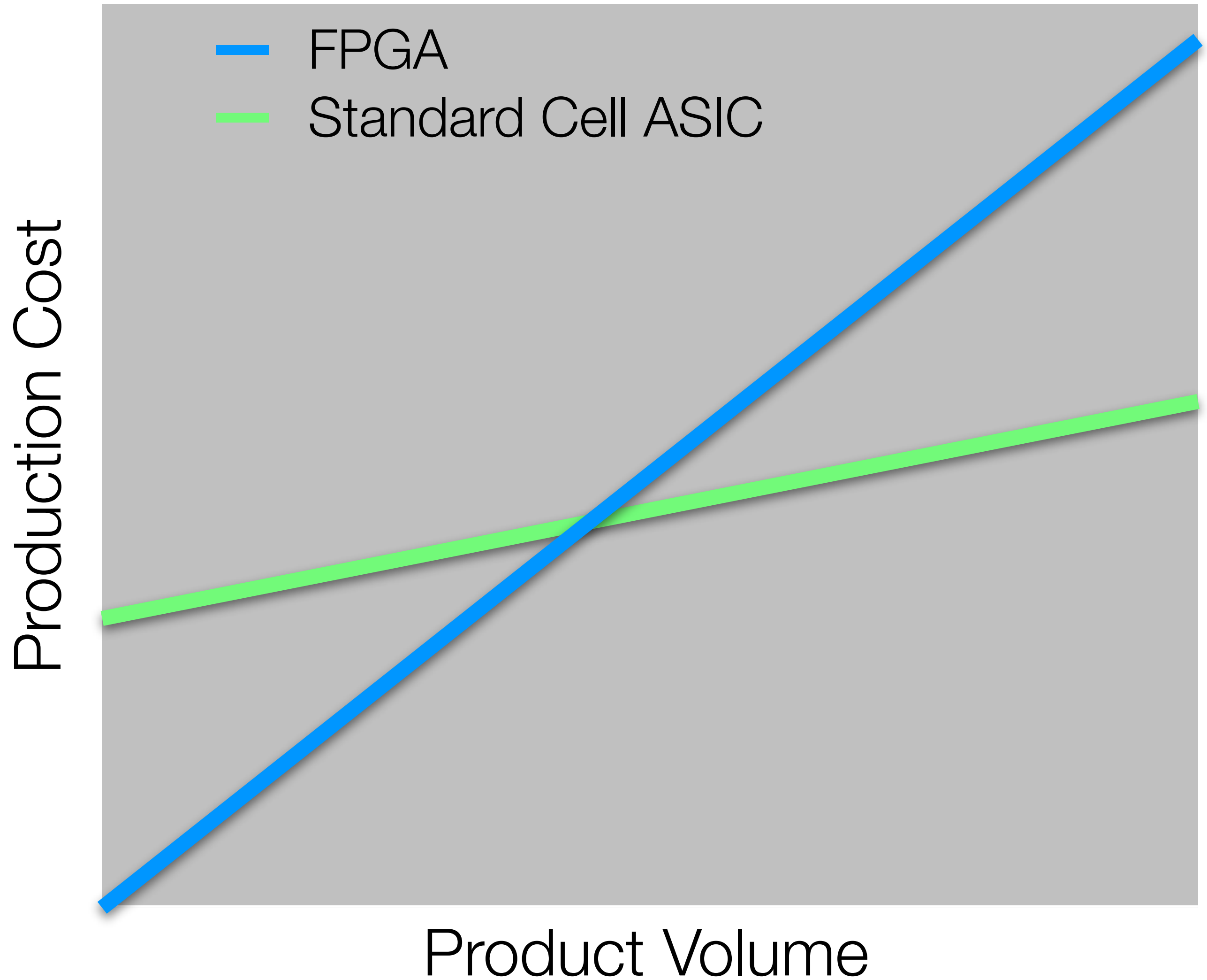
January 11, 2007

Declining ASIC Starts



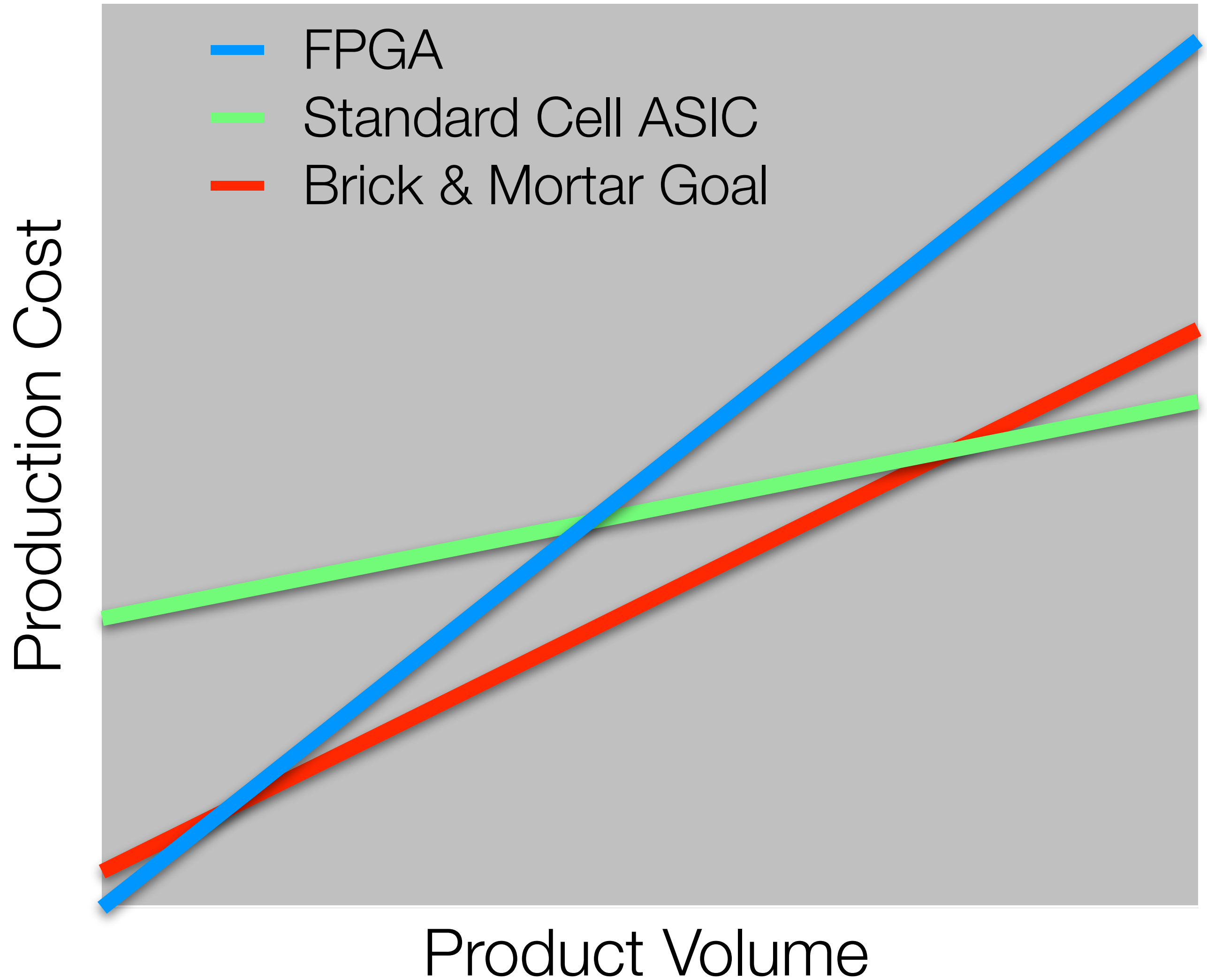
[DataQuest]

Cost of Production



[www.edn.com] 3

Cost of Production

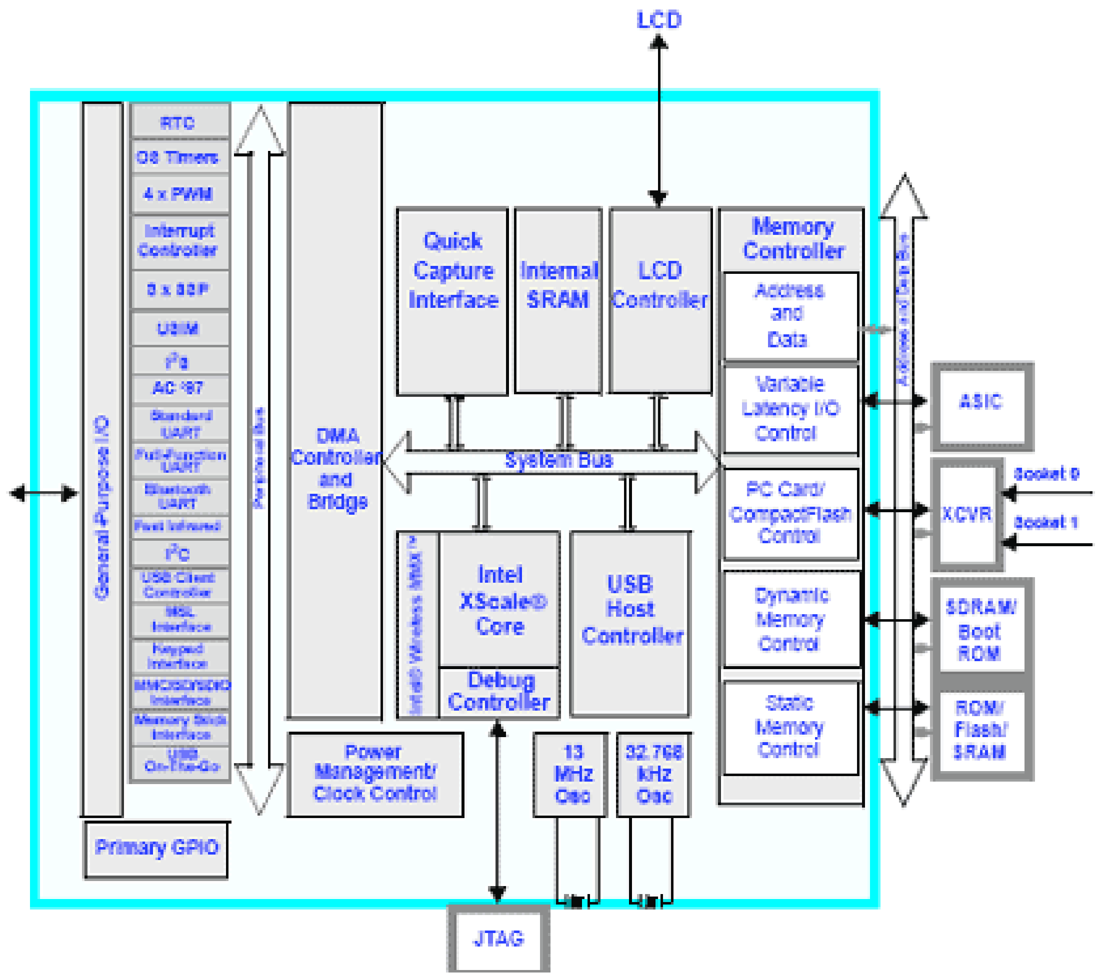


[www.edn.com] [4]

System on Chip

- Assemble system out of pre-designed components
- Reduce design time
  - *In 2004, one engineer costed $392,000 annually [www.design-reuse.com]*
- Minimize bugs
  - *Initial bugs can cost 50% of revenue [www.design-reuse.com]*

PXA27X processor

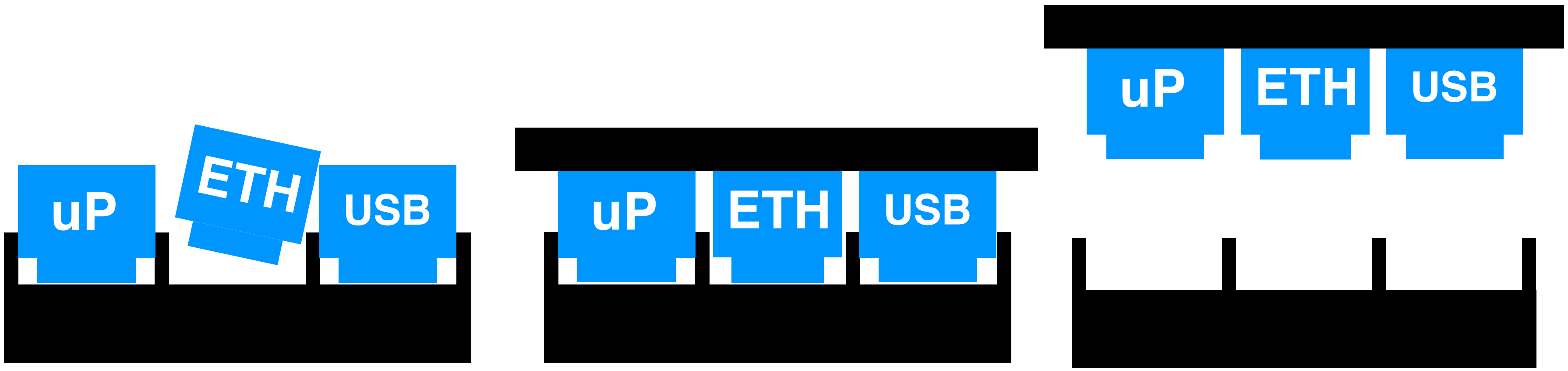


[www.tomshardware.co.uk]

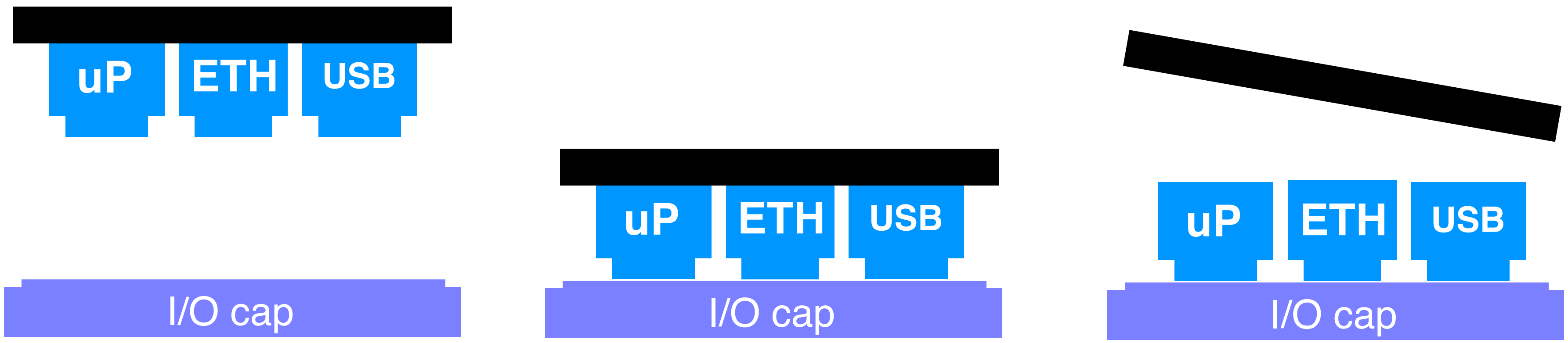
Brick and Mortar: Assembly

- Bricks -- ASIC chips
  - standard interface
  - implement standard functions
    - i.e., USB, VGA controller, ethernet NIC, PCI bridge, DMA, SRAM, 3DES, JPEG codec, RISC core

Brick and Mortar: Assembly

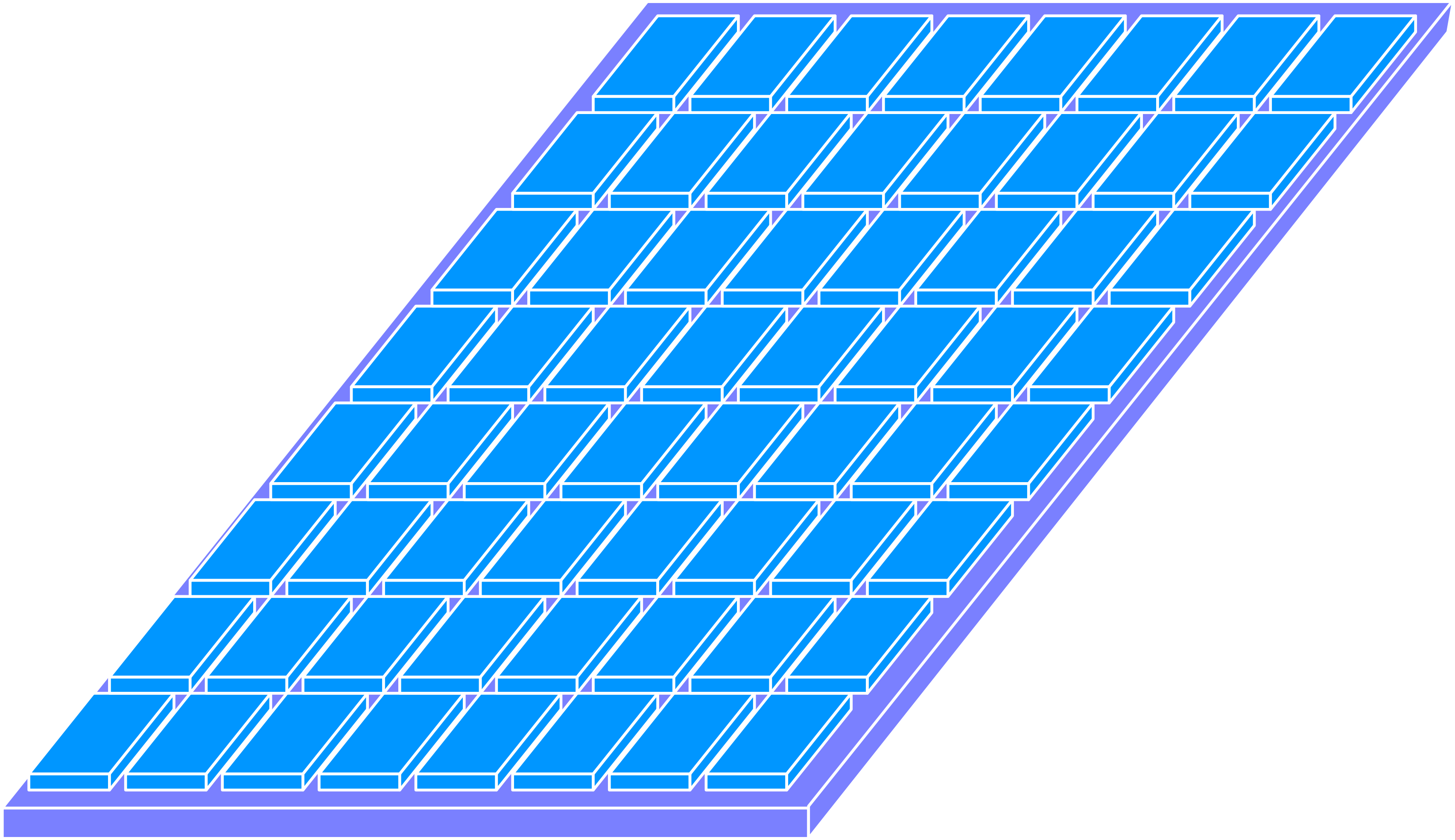


Brick and Mortar: Assembly



8

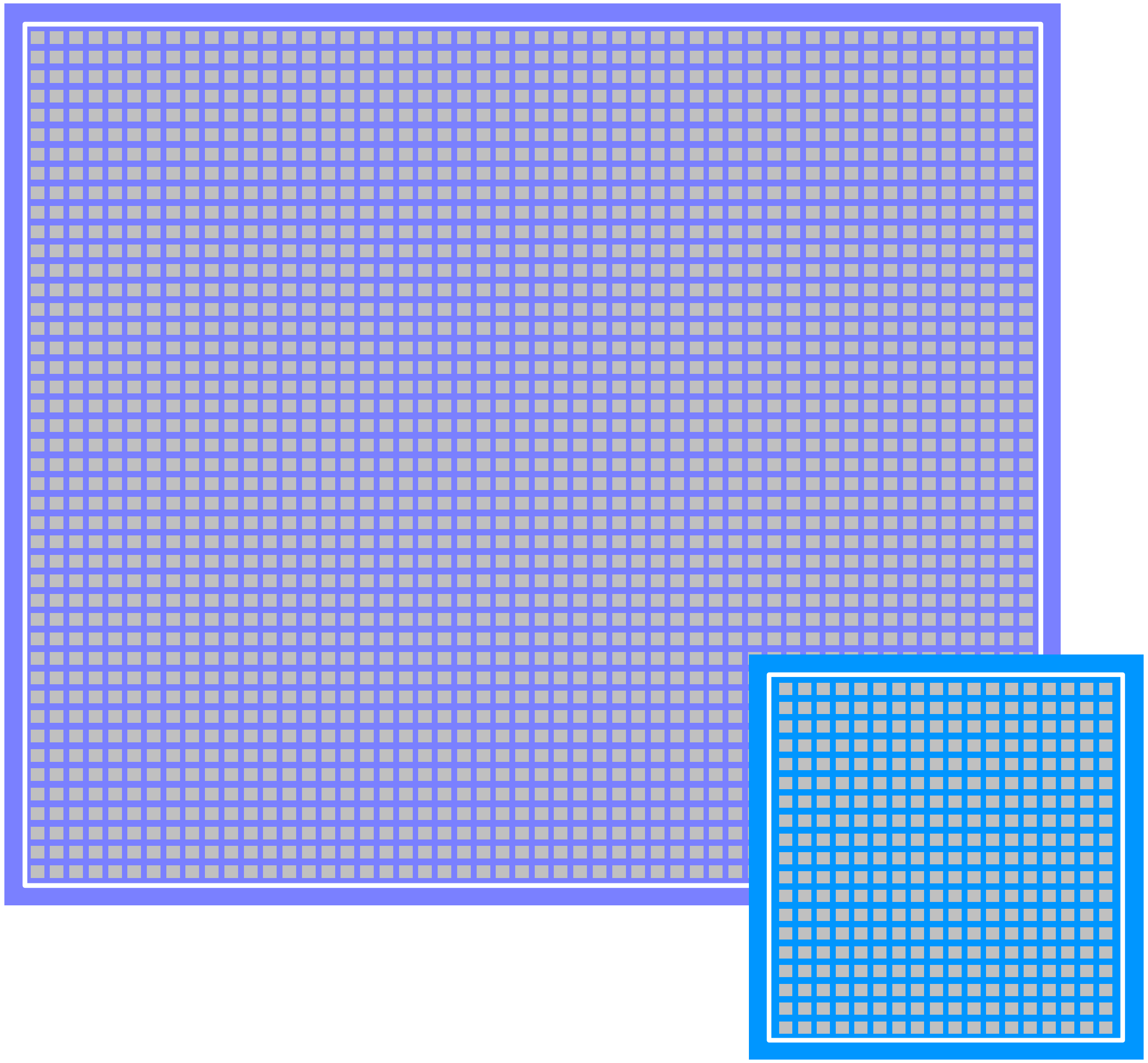
Brick and Mortar: Chip



9

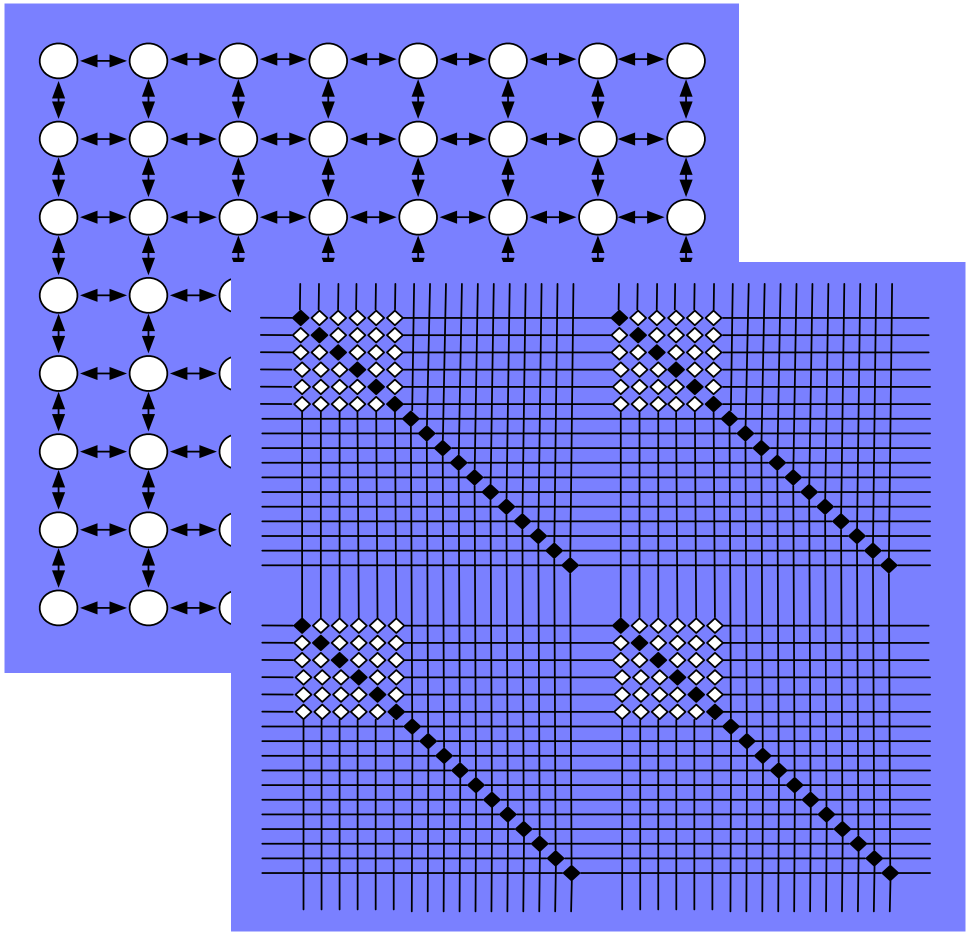
Brick and Mortar: I/O Pads

- One surface covered with I/O pads
  - 25 um x 25 um / pad
  - 2.5 Gbps / pad

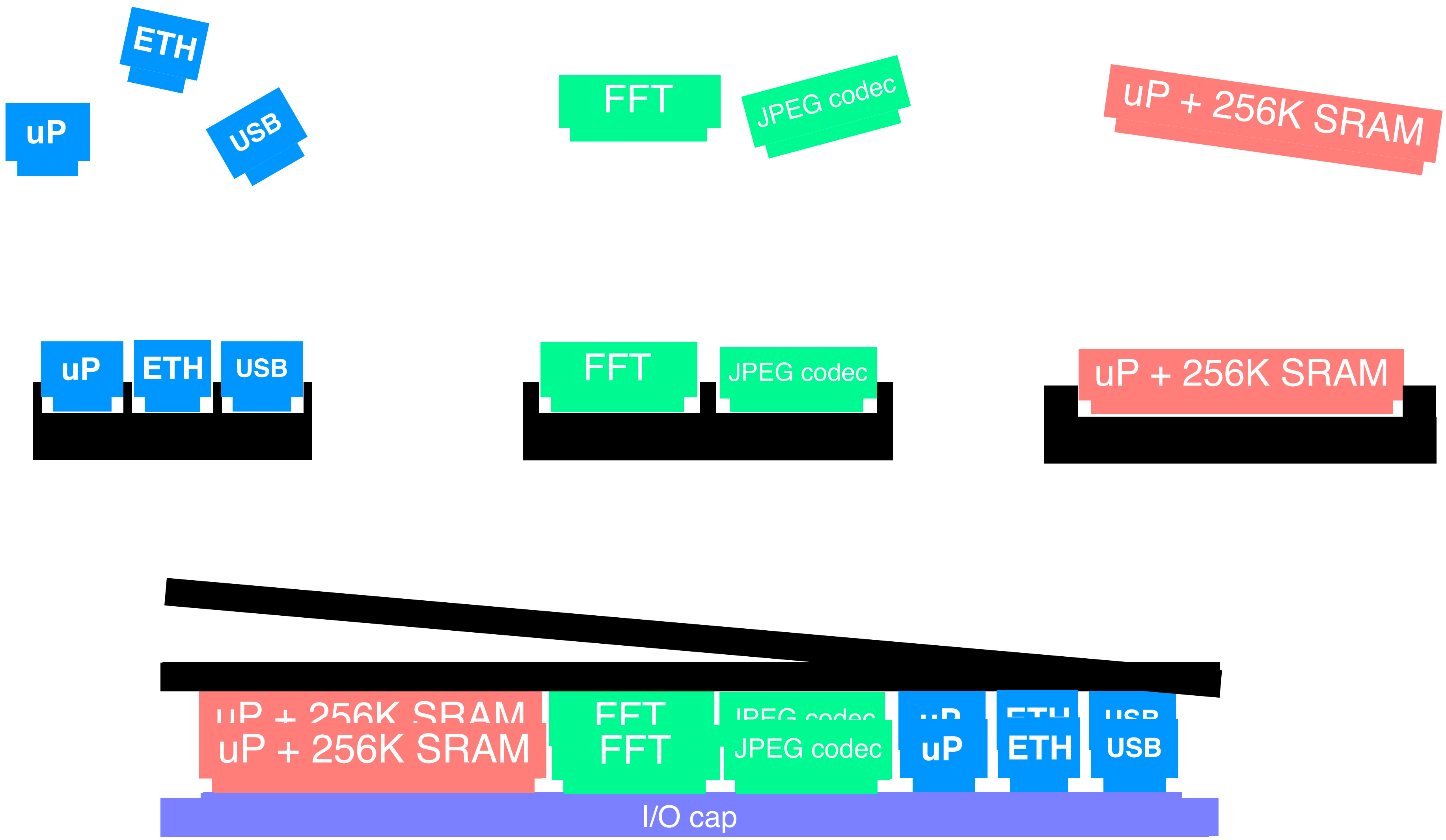


Brick and Mortar: I/O Cap Interconnect

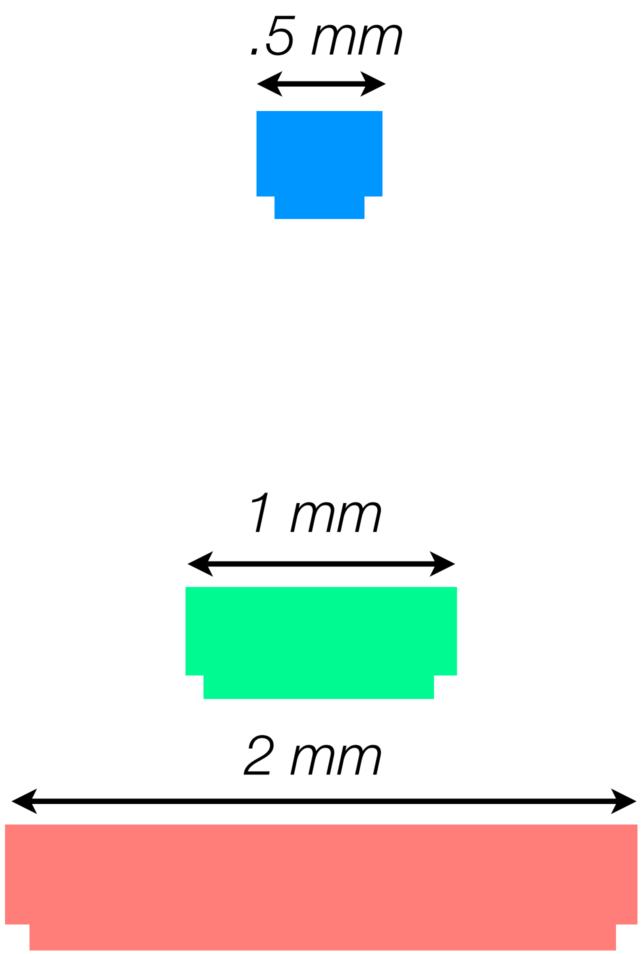
- I/O cap -- ASIC chip implementing inter-brick interconnect
  - packet-switched network
  - FPGA-like, island style configurable interconnect



Brick and Mortar: Multiple Brick Sizes



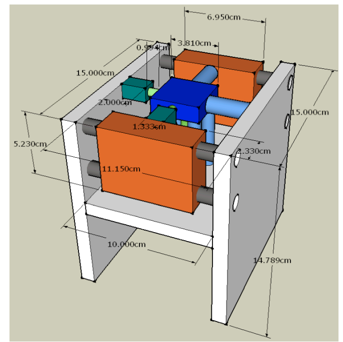
Brick and Mortar: Multiple Brick Sizes

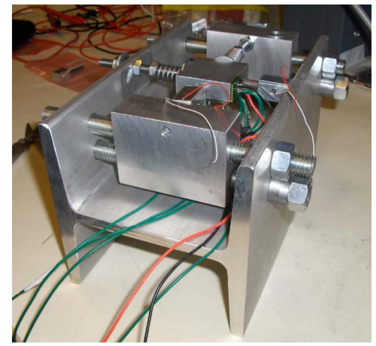


Function	Cite	Circuit Area (um^2)	Max. Circuit Freq. (MHz)	Min. Perf. (Mbps)	**0.25** mm² brick	**1.0** mm² brick	**4.0** mm² brick
					Valid Freq. Range (MHz)		
Small Bricks							
USB 1.1 PHYSICAL LAYER	[33]	2,201	2941	12	**2 - 2941**	No benefit	No benefit
VITERBI	[45]	2,614	1961	-	**N/A - 1961**	No benefit	No benefit
VGA/LCD CONTROLLER	[33]	4,301	1219	-	**N/A - 1046**	N/A -1219	No benefit
WB DMA	[33]	13,684	1163	-	**N/A - 521**	N/A - 1163	No benefit
MEMORY CONTROLLER	[33]	29,338	952	-	**N/A - 843**	N/A - 952	No benefit
TRI MODE ETHERNET	[33]	32,009	893	1000	**125 - 893**	No benefit	No benefit
PCI BRIDGE	[33]	76,905	1042	-	**N/A - 610**	N/A - 1042	No benefit
WB Switch (8 master, 16 slave)	[33]	81,073	1087	-	**N/A - 88**	N/A - 353	N/A - 1087
FPU	[33]	85,250	1515	-	**N/A - 505**	N/A - 1515	No benefit
DES	[33]	85,758	1370	1000	**16 - 1203**	16 - 1370	No benefit
16K SRAM (Singleport)	[6]	195,360	2481	-	**N/A - 2481**	No benefit	No benefit
AHO-CORASIK STR. MATCH	[50]	201,553	2481	-	**N/A - 1331**	N/A - 2481	No benefit
RISC CORE (NO FPU) / 8K CACHE	[33] [6]	219,971	1087	-	**N/A - 1087**	No benefit	No benefit
8K SRAM (Dualport)	[6]	230,580	1988	-	**N/A - 1988**	No benefit	No benefit
Medium Bricks							
TRIPLE DES	[33]	294,075	1282	1000	No space	**16 - 1282**	No benefit
FFT	[44]	390,145	1220	-	No space	**N/A - 1220**	No benefit
JPEG DECODER	[33]	625,457	629	-	No space	**N/A - 629**	No benefit
64K SRAM (Singleport)	[6]	682,336	2315	-	No space	**N/A - 2315**	No benefit
32K SRAM (Dualport)	[6]	733,954	1842	-	No space	**N/A - 1842**	No benefit
RISC CORE + 64K CACHE	[33] [6]	864,017	1087	-	No space	**N/A - 1087**	No benefit
Large Bricks							
256K SRAM (Singleport)	[6]	2,729,344	2315	-	No space	No space	**N/A - 2315**
128K SRAM (Dualport)	[6]	2,935,817	2882	-	No space	No space	**N/A - 2882**
RISC CORE + 256K CACHE	[33] [6]	3,111,025	1087	-	No space	No space	**N/A - 1087**

Advantages of Brick and Mortar

- Low manufacturing costs
  - *no custom masks*
  - *small design & verification costs*
  - *low-cost assembly system (fluidic self assembly)*
- ASIC-like degree of circuit integration
- Heterogeneous processes for bricks
- Exclude defective components from assembly
- Leverage process variation for high performance designs

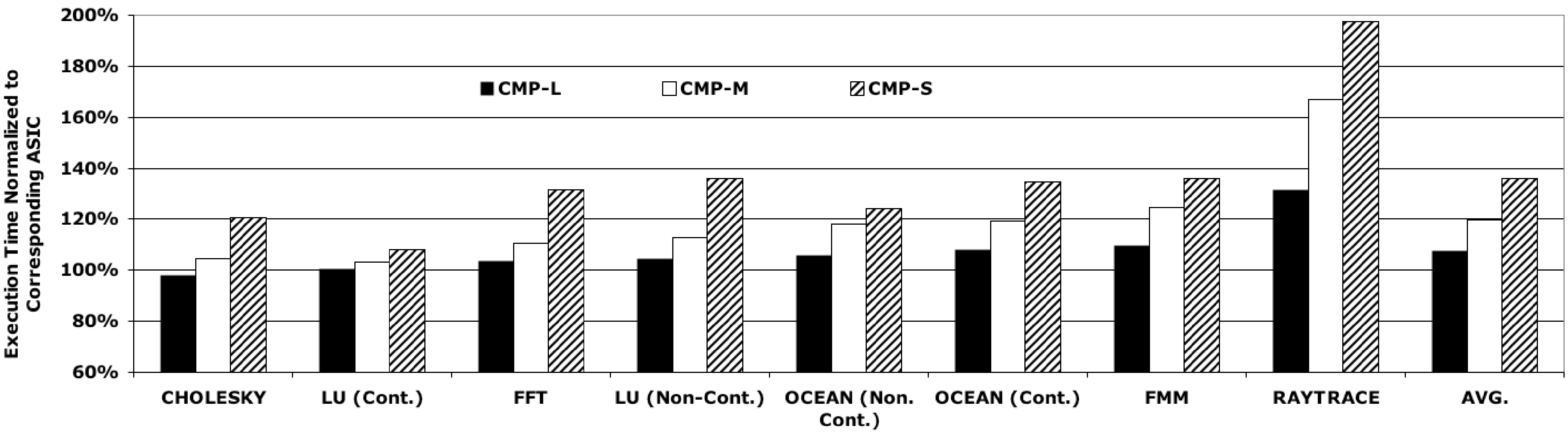




14

Preliminary Performance Analysis

- Three, 16-way CMP designs
- Only 8% - 36% slowdown relative to ASIC



15

Why RAMP?

- Once a design has been tested and validated on RAMP platform
  - Less costly, per unit, than FPGAs (or boards)
  - Higher-speed than FPGAs

Conclusion

- Systems built out of ASIC bricks bonded to an interconnect ASIC
- A viable, low-cost technology if properly architected:
  - appropriate brick functions
  - general, flexible interconnect
  - efficient inter-ASIC communication

17

Questions & Discussion