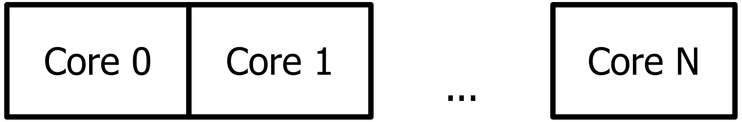
BlackParrot Deep Dive

BlackParrot Project Goals

Become the default open source Linux-capable RISC-V multicore used by the planet.



Create a HW development style that reconciles modern software engineering with the challenging requirements of hardware.

Approved for public release: distribution unlimited.

The BlackParrot “Genesis Release” Team

Prof. Michael Taylor

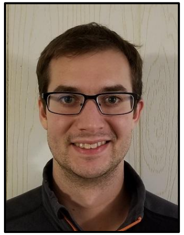
BOSTON UNIVERSITY

Prof. Ajay Joshi

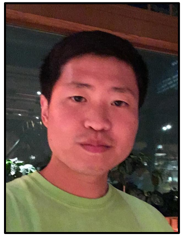
Prof. Mark Oskin

Dan Petrisko

Farzam Gilani



Mark Wyse



Tommy Jung

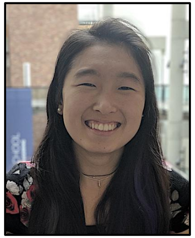
Scott Davidson

Leila Delshad

Boyou Zhou

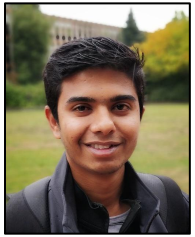
Zahra Azad

Furkan Eris

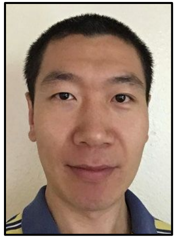


Katie Lim

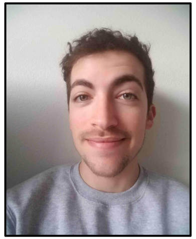
Yongqin Wang



Bandhav Veluri

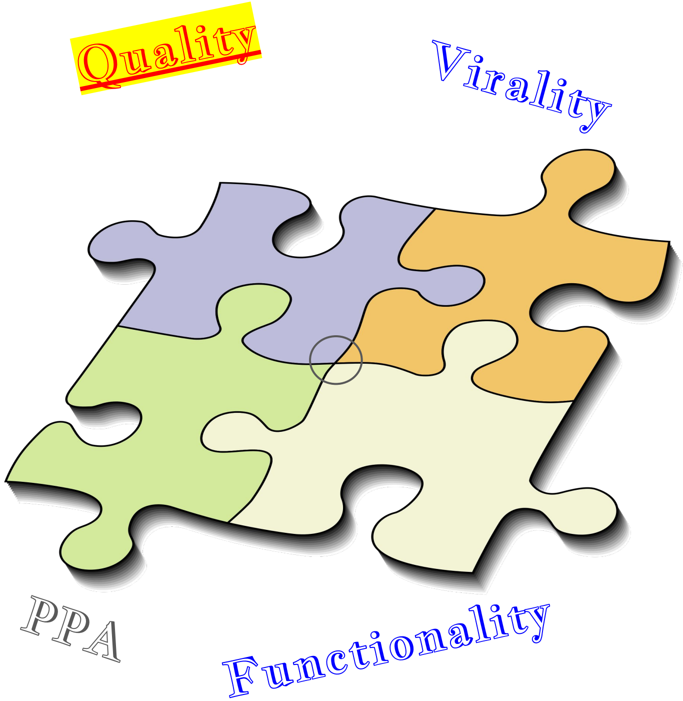


Chun Zhao



Tavio Guarino

BlackParrot: Four Success Metrics *(achieve these and BlackParrot will become the Linux of RISC-V)*



Approved for public release: distribution unlimited.

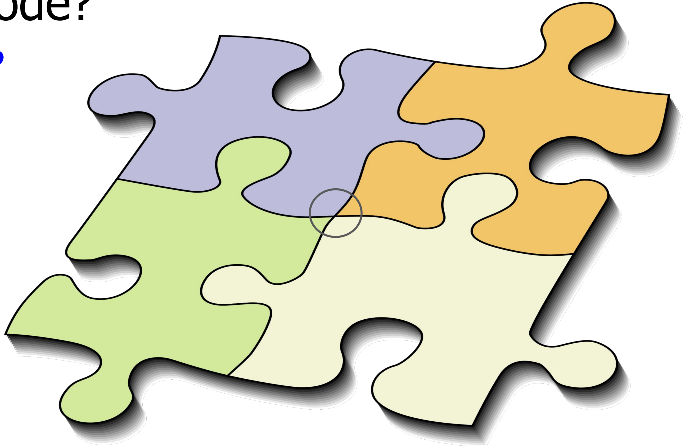
BlackParrot: Four Success Metrics *(achieve these and BlackParrot will become the Linux of RISC-V)*

Quality

Virality

Will People Trust Our Code?

- Is it easy to understand?
- Is it secure?
- Is it validated?
- Will you put it in Silicon?



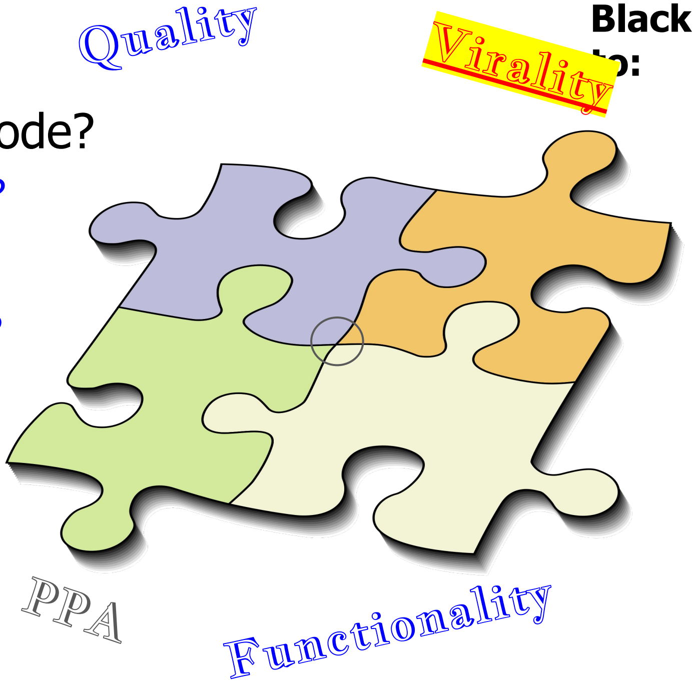
PPA

Functionality

BlackParrot: Four Success Metrics *(achieve these and BlackParrot will become the Linux of RISC-V)*

Will People Trust Our Code?

Is it easy to understand?
Is it secure?
Is it validated?
Will you put it in Silicon?



BlackParrot is a “stone soup” designed to:

convince the smartest people in the world to improve it.

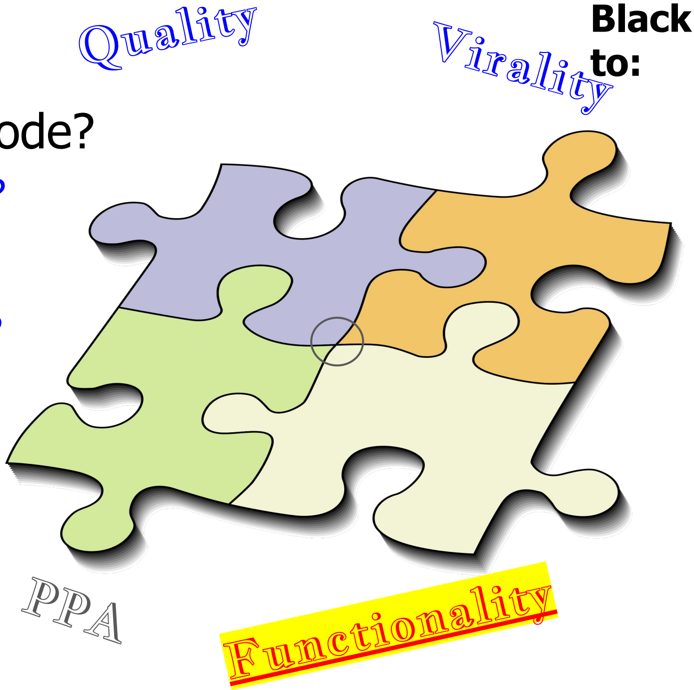
scale to many users.

get companies to invest and become stewards of the code.

BlackParrot: Four Success Metrics *(achieve these and BlackParrot will become the Linux of RISC-V)*

Will People Trust Our Code?

Is it easy to understand?
Is it secure?
Is it validated?
Will you put it in Silicon?



BlackParrot is a "stone soup" designed to:

convince the smartest people in the world to improve it.

scale to many users.

get companies to invest and become stewards of the code.

Does the code have the features people need?

And leave out the ones they don't?

Approved for public release: distribution unlimited.

BlackParrot: Four Success Metrics *(achieve these and BlackParrot will become the Linux of RISC-V)*

Quality

Will People Trust Our Code?

Is it easy to understand?
Is it secure?
Is it validated?
Will you put it in Silicon?

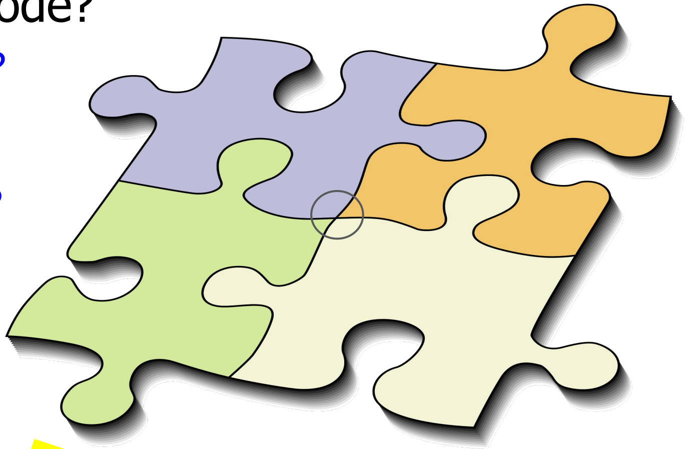
Virality

BlackParrot is a "stone soup" designed to:

convince the smartest people in the world to improve it.

scale to many users.

get companies to invest and become stewards of the code.



PPA

Is the code Pareto optimal in terms of Power, Performance, and Area?

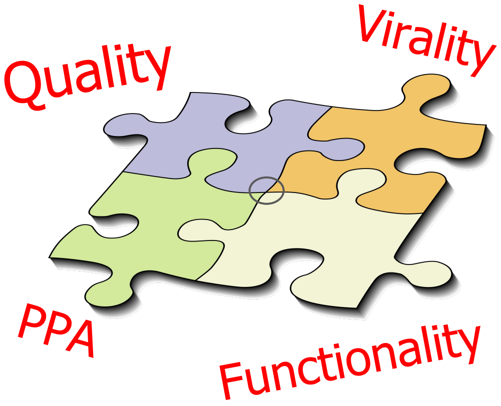
Functionality

Does the code have the features people need?

And leave out the ones they don't?

The BlackParrot Manifesto

- BE TINY
  - Place a premium on a small, understandable, agile, secure code base.
  - Minimize unused features or configurations that increase complexity and verification.
- BE MODULAR
  - Use well-defined interfaces that enable scalable, global participation.
  - Enable modular testability & CI.
- BE FRIENDLY
  - Welcome contributions and distributed ownership.
  - Combat "Not Invented Here" Syndrome.
  - Be easy to use.



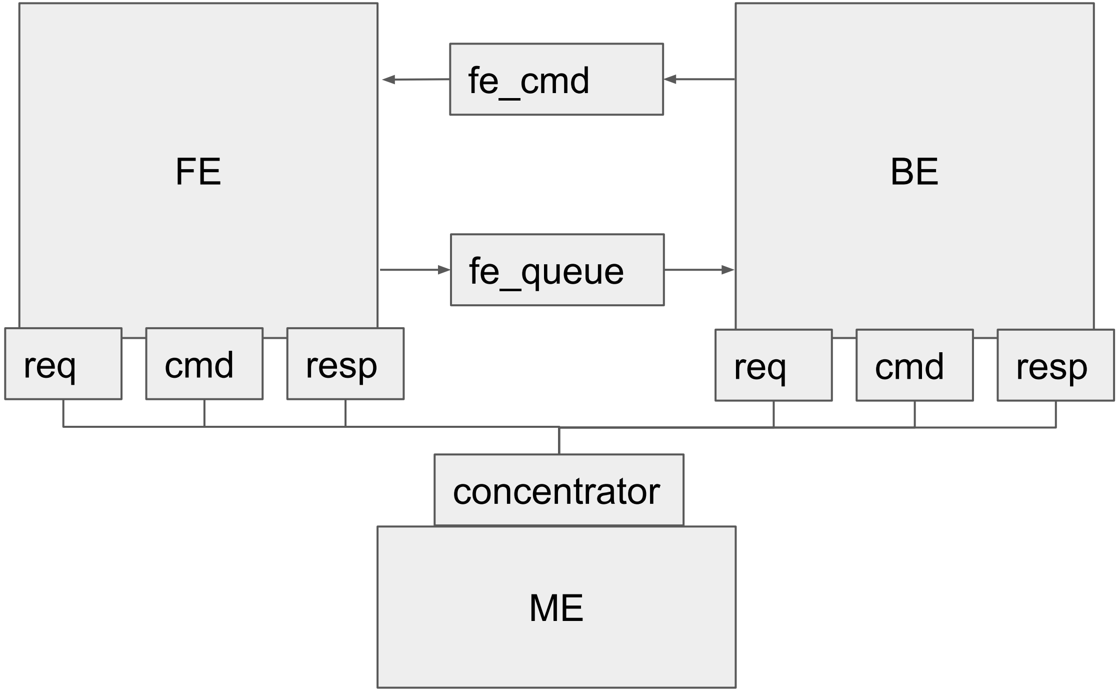
Source: http://www.parrotbreeding.com.au/wp-content/uploads/2009/12/yellow-tailed-black-cockatoo-female.jpg

Focus on interfaces, not implementation

- Borrowing from software, we focus on defining clean and narrow interfaces
- Then microarchitectural change only needs to be verified at the module level, rather than the system level
  - E.g. Adding a new branch predictor only affects the Front End, not the Back End
- Challenge is to make these interfaces flexible enough to support various levels of sophistication in implementation without incurring hardware overhead

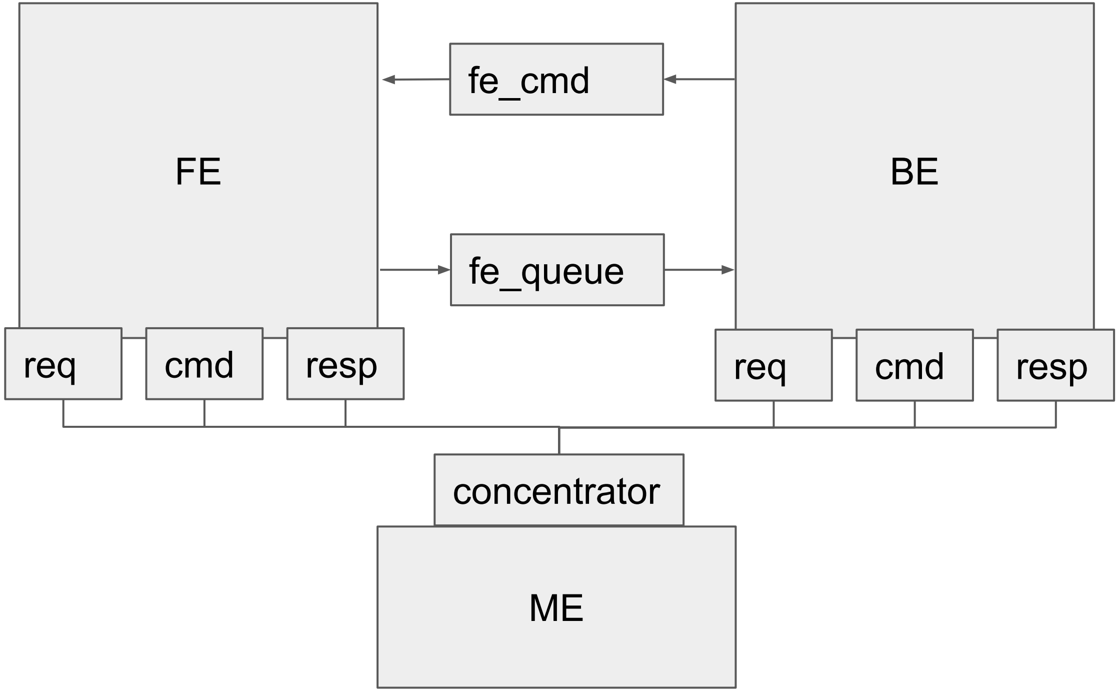
Decoupled Core Uarch

- Front End (FE)
  - Fully speculative region
  - No architectural state lives here
  - Supplies Back End with pc/instruction pairs
- Back End (BE)
  - Executes RISC-V instructions
  - Manages virtual memory/privilege levels/interrupts
- Memory End (ME)
  - Maintains coherence
  - Manages directory tags



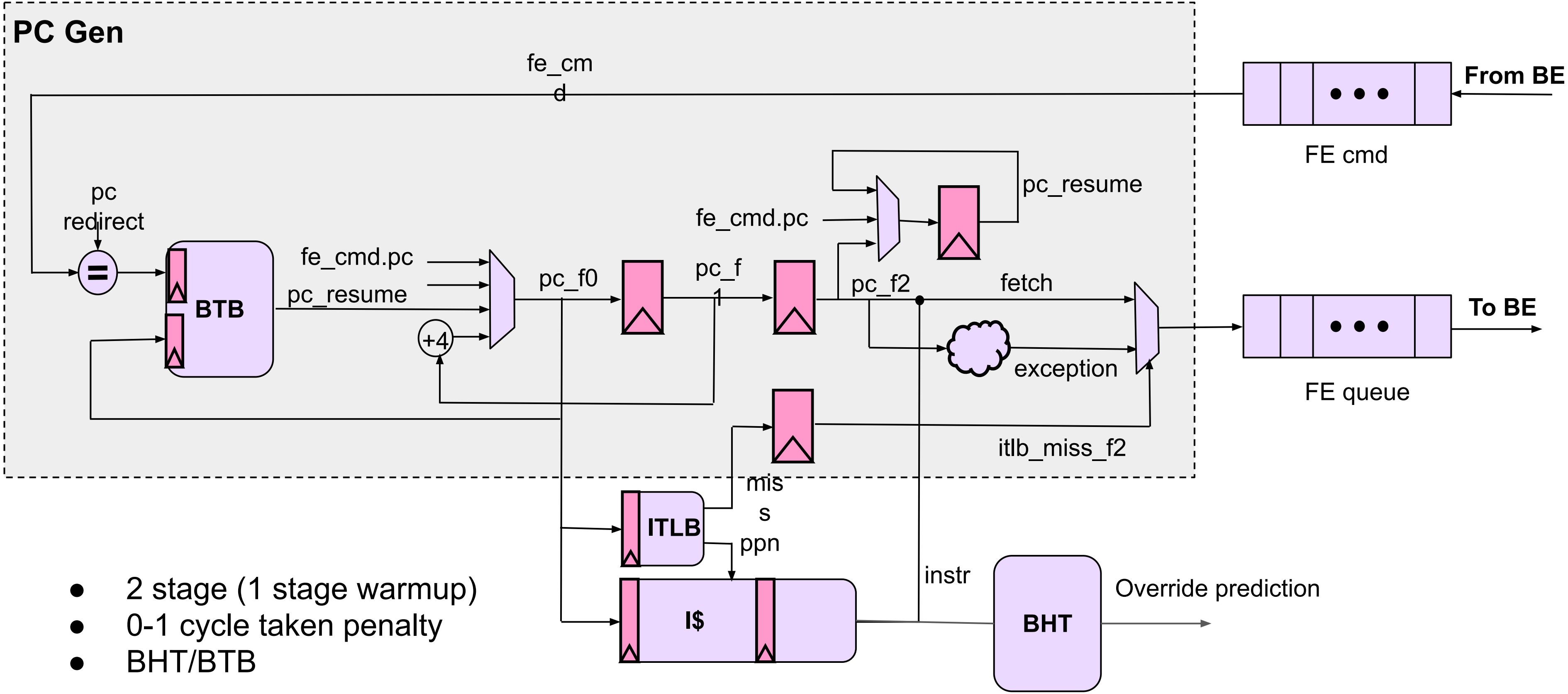
Interfaces

- fe_queue
  - pc/instr pairs
  - pc/exception pairs
    - Memory access faults
    - tlb miss
- fe_cmd
  - pc redirections
    - mispredict
    - trap
  - tlb mappings, etc.
- lce_req
  - Request from the LCE to the CCE (I have a cache miss)
- lce_cmd
  - Command to LCE (set tag and data to x)
- lce_resp
  - Response to a command from CCE (I have evicted a line, here is the dirty data)



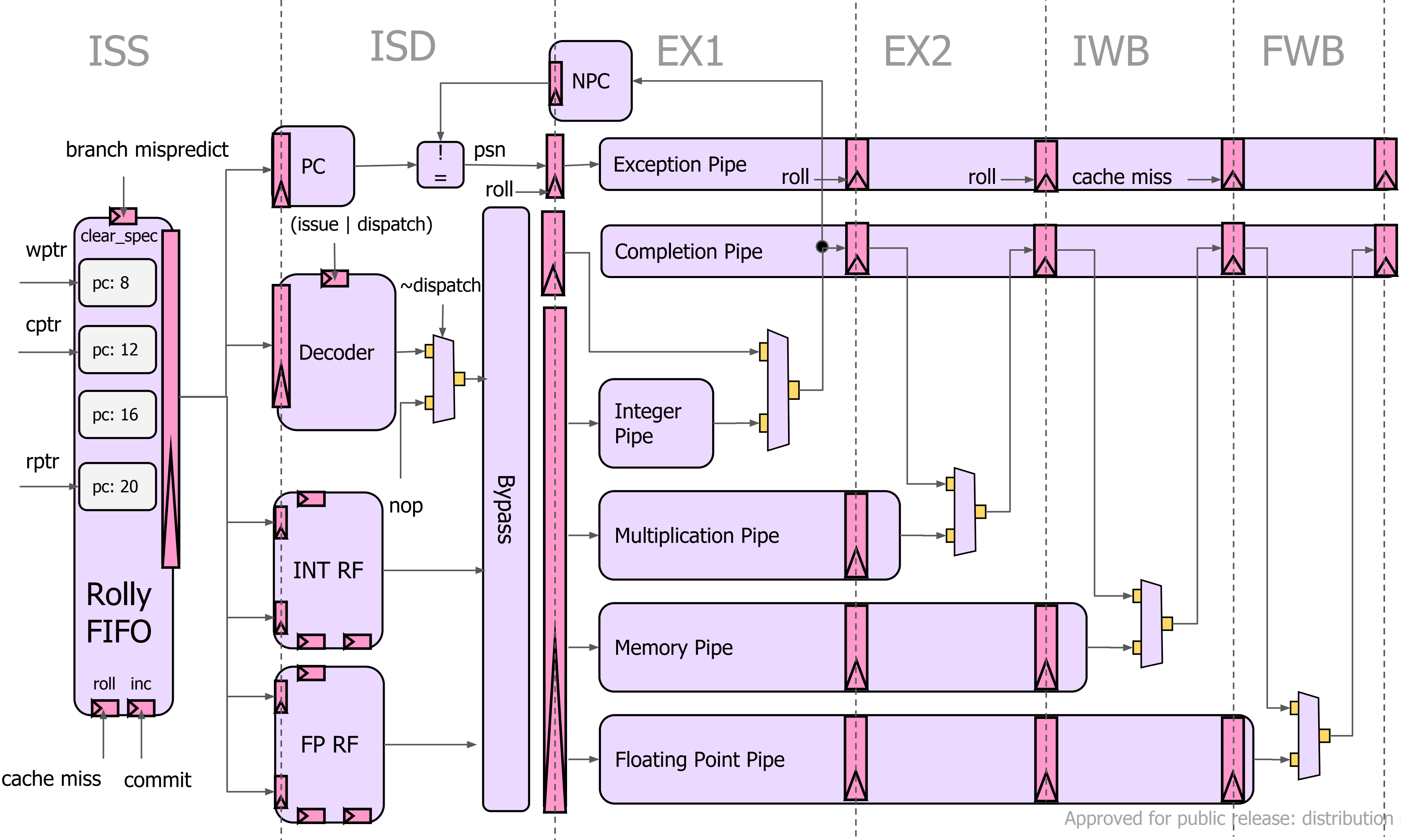
Current Core Implementation

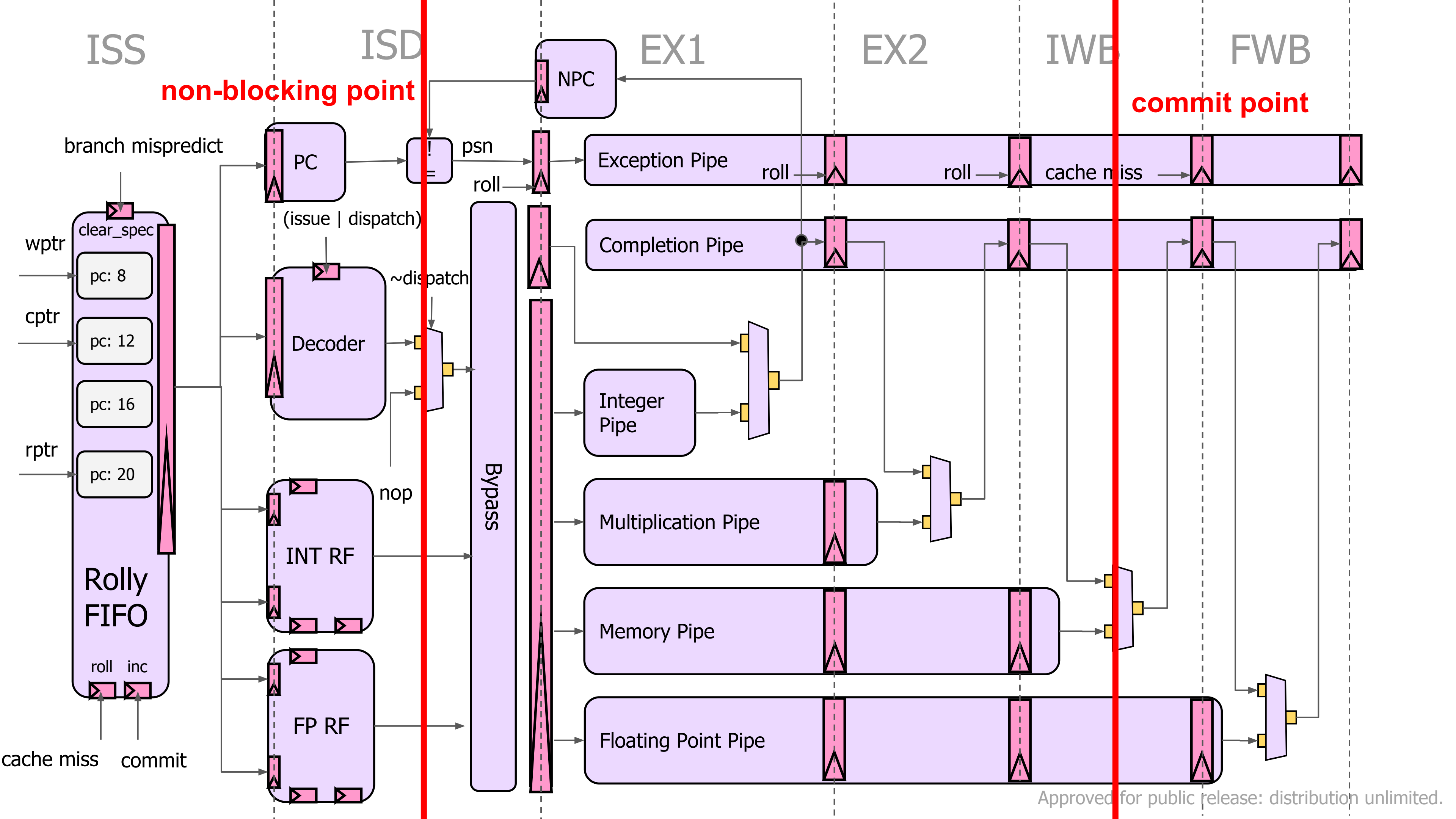
BlackParrot Front End



- 2 stage (1 stage warmup)
- 0-1 cycle taken penalty
- BHT/BTB

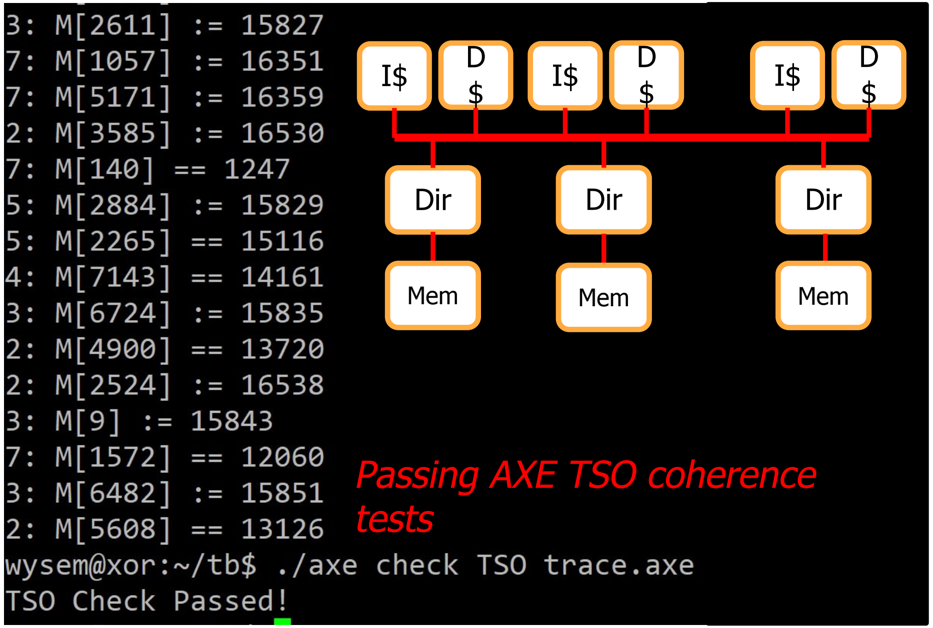
BlackParrot BE




ISS
ISD
EX1
EX2
IWB
FWB
non-blocking point
commit point
NPC
branch mispredict
PC
psn
roll
!
=
(issue | dispatch)
Exception Pipe
roll
roll
cache miss
Completion Pipe
clear_spec
wptr
cptr
rptr
pc: 8
pc: 12
pc: 16
pc: 20
Rolly FIFO
roll
inc
cache miss
commit
Decoder
~dispatch
nop
INT RF
FP RF
Bypass
Integer Pipe
Multiplication Pipe
Memory Pipe
Floating Point Pipe

POSH: U. Washington’s BlackParrot Programmable Cache Coherence

First open-source programmable directory-based coherence controller



```
3: M[2611] := 15827
7: M[1057] := 16351
7: M[5171] := 16359
2: M[3585] := 16530
7: M[140] == 1247
5: M[2884] := 15829
5: M[2265] == 15116
4: M[7143] == 14161
3: M[6724] := 15835
2: M[4900] == 13720
2: M[2524] := 16538
3: M[9] := 15843
7: M[1572] == 12060
3: M[6482] := 15851
2: M[5608] == 13126
wysem@xor:~/tb$ ./axe check TSO trace.axe
TSO Check Passed!
```

Passing AXE TSO coherence tests

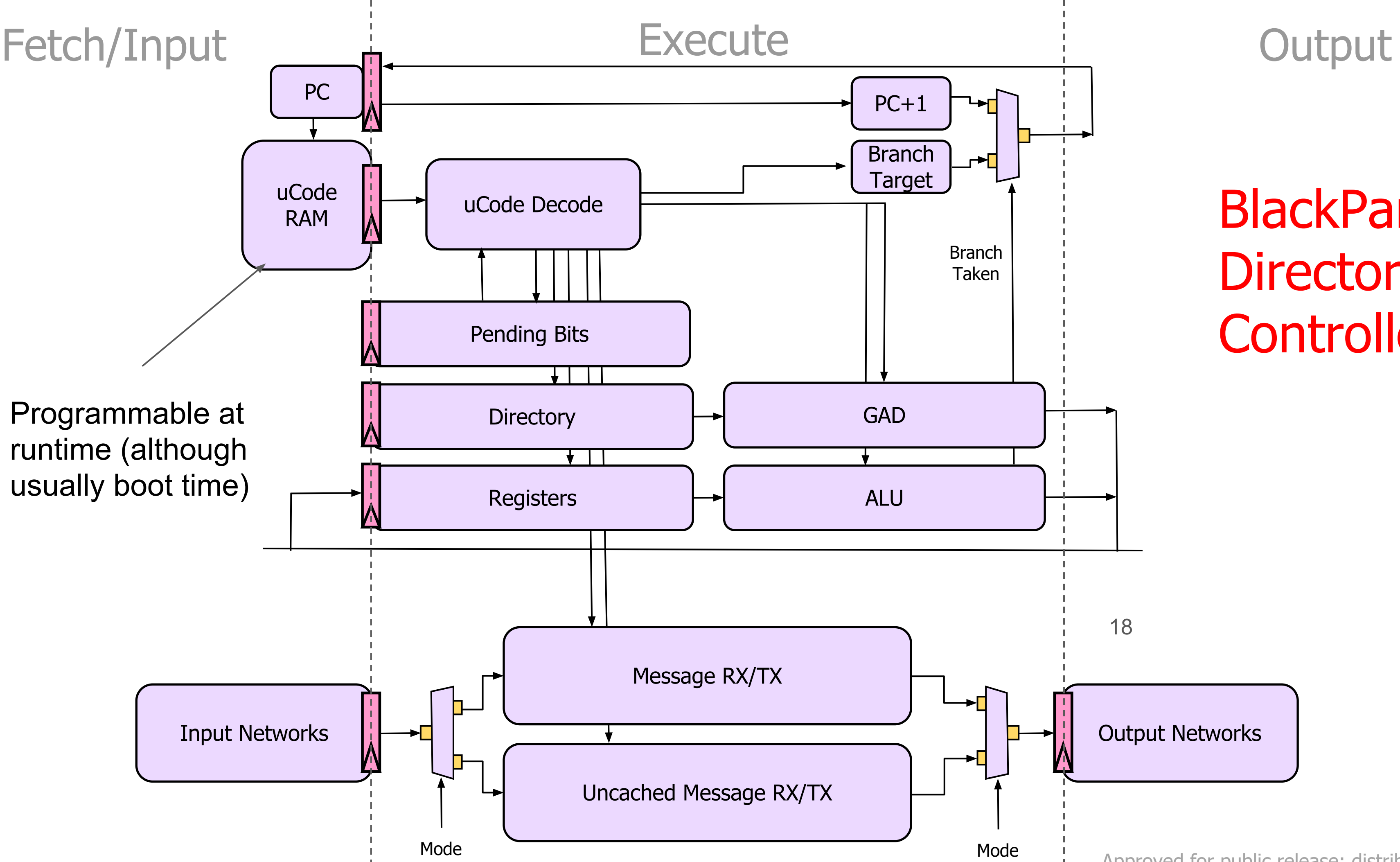
Major Accomplishments to date:

- First open-source *programmable* directory-based cache coherence controller
- First open-source *race-free-by-design* directory-based coherence protocol implementation
- First open-source synthesizable design for exploring interplay of cache coherence and security

Challenges:

- Directory sharding leads to excessively wide SRAMS
    - Solved by modifying coherence engine to support configurable sequential readout of tags

BlackParrot Directory Controller



Programmable at runtime (although usually boot time)

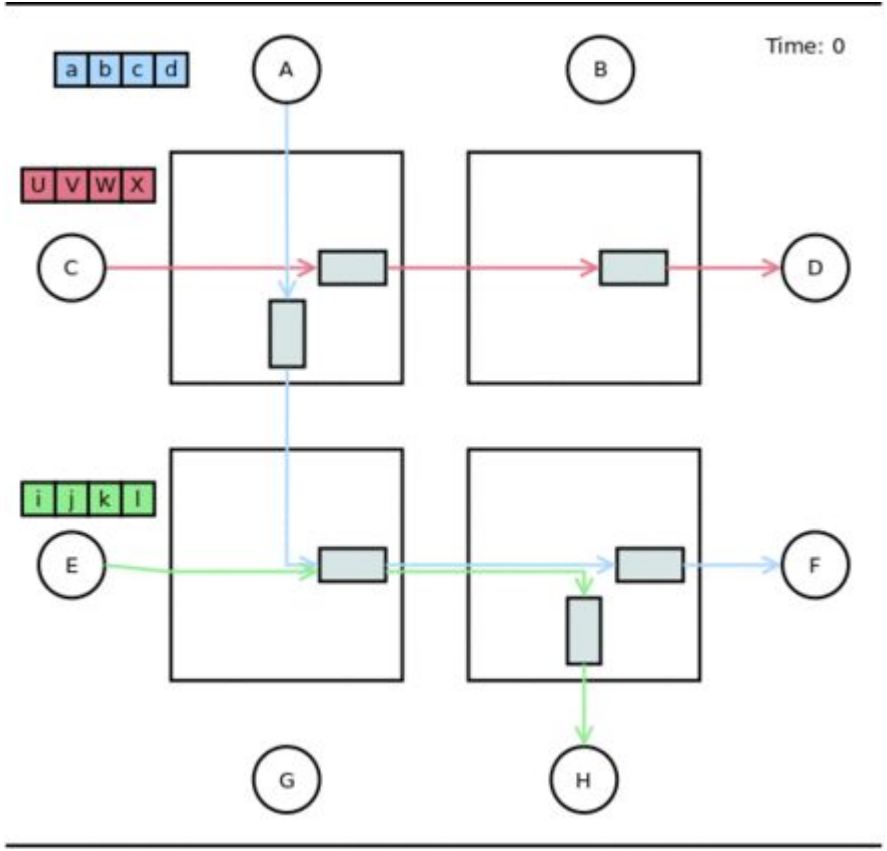
Feature wishlist

- Better / more flexible branch prediction (FE)
- More RISC-V features (FP, multiplication) (BE)
- Cool CCE features (prefetching, alternate coherence protocols)
- More flexible configurations (cache size, VM/no VM, etc.)

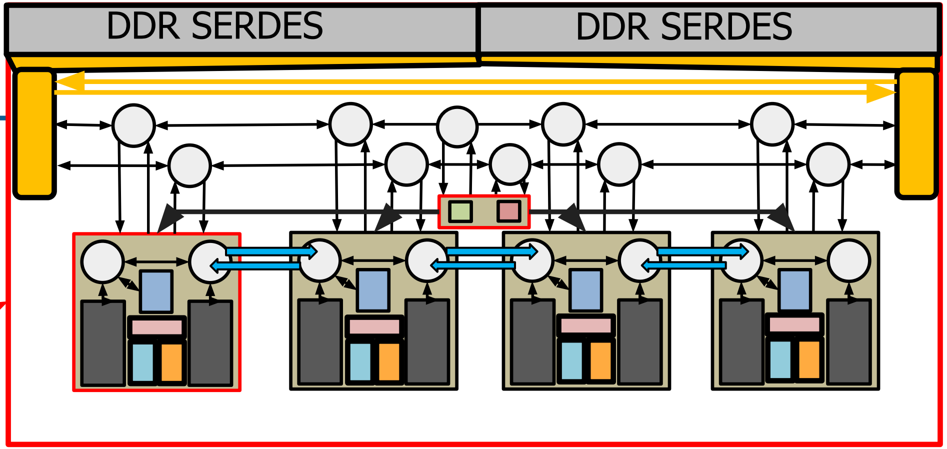
Current System Implementation

Wormhole routing

- Scalable and flexible routing strategy
- Smaller link widths than single flit routing
- Serialization / deserialization penalty
- High network occupancy
- Highly deadlock prone (but we avoid it by construction)
- Could other strategies do better?



BlackParrot One ASIC



Taped out July 13, 2019!
GF 12nm Process Technology
3mm x 3mm die

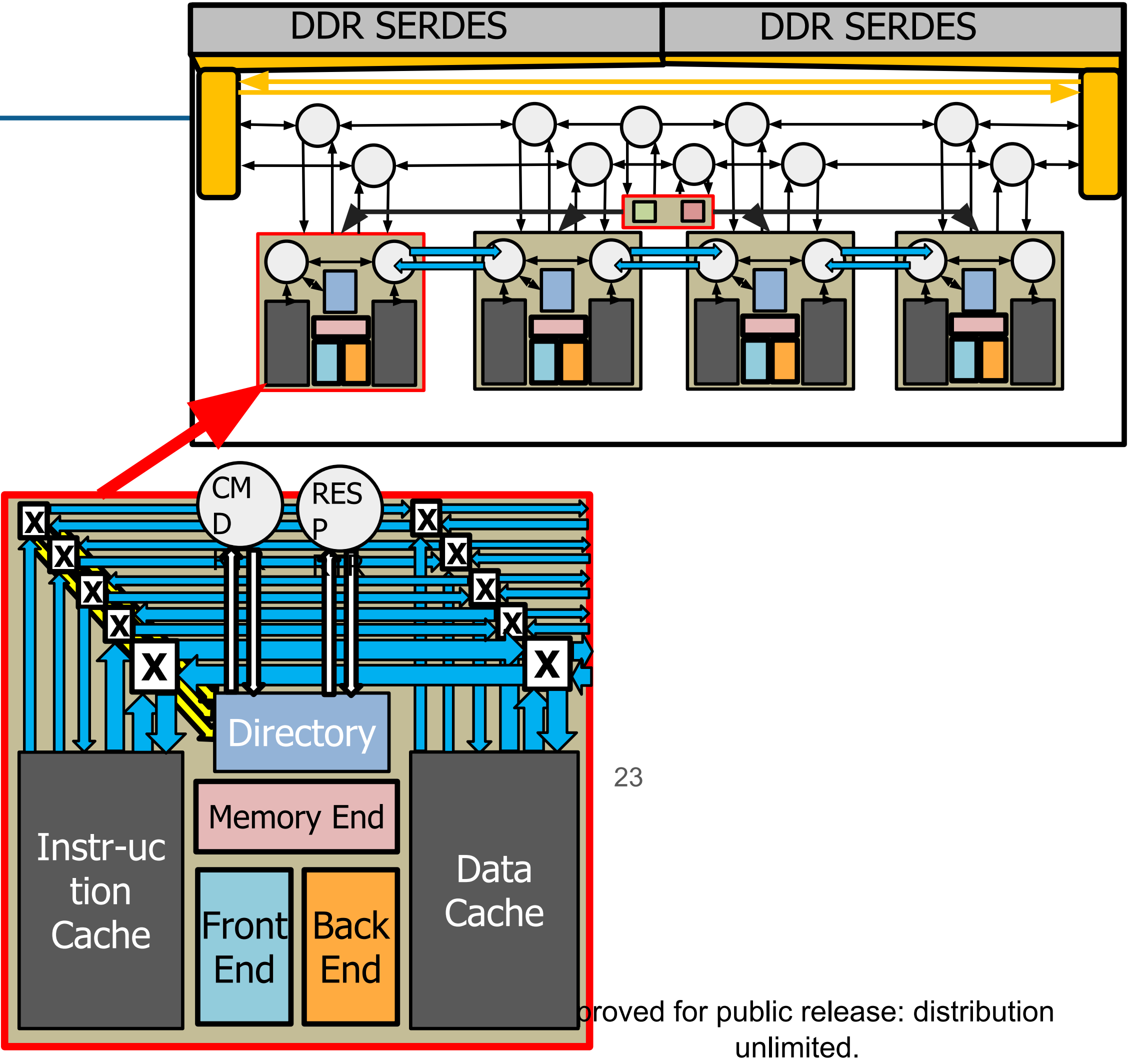
4-core 64b RISC-V multicore
CLINT Interrupt Controller
Off-chip SERDES for DRAM and I/O
Chips can be chained

BlackParrot One ASIC

Taped out July 13, 2019!
GF 12nm Process Technology
3mm x 3mm die

4-core 64b RISC-V multicore
CLINT Interrupt Controller
Off-chip SERDES for DRAM and I/O
Chips can be chained

Each core:
- RV64IA with Virtual Memory
- Single-issue In-order
- 32K Data cache
- 32K Instruction cache
- 64-entry BTB
- 8-entry DTLB
- 8-entry ITLB



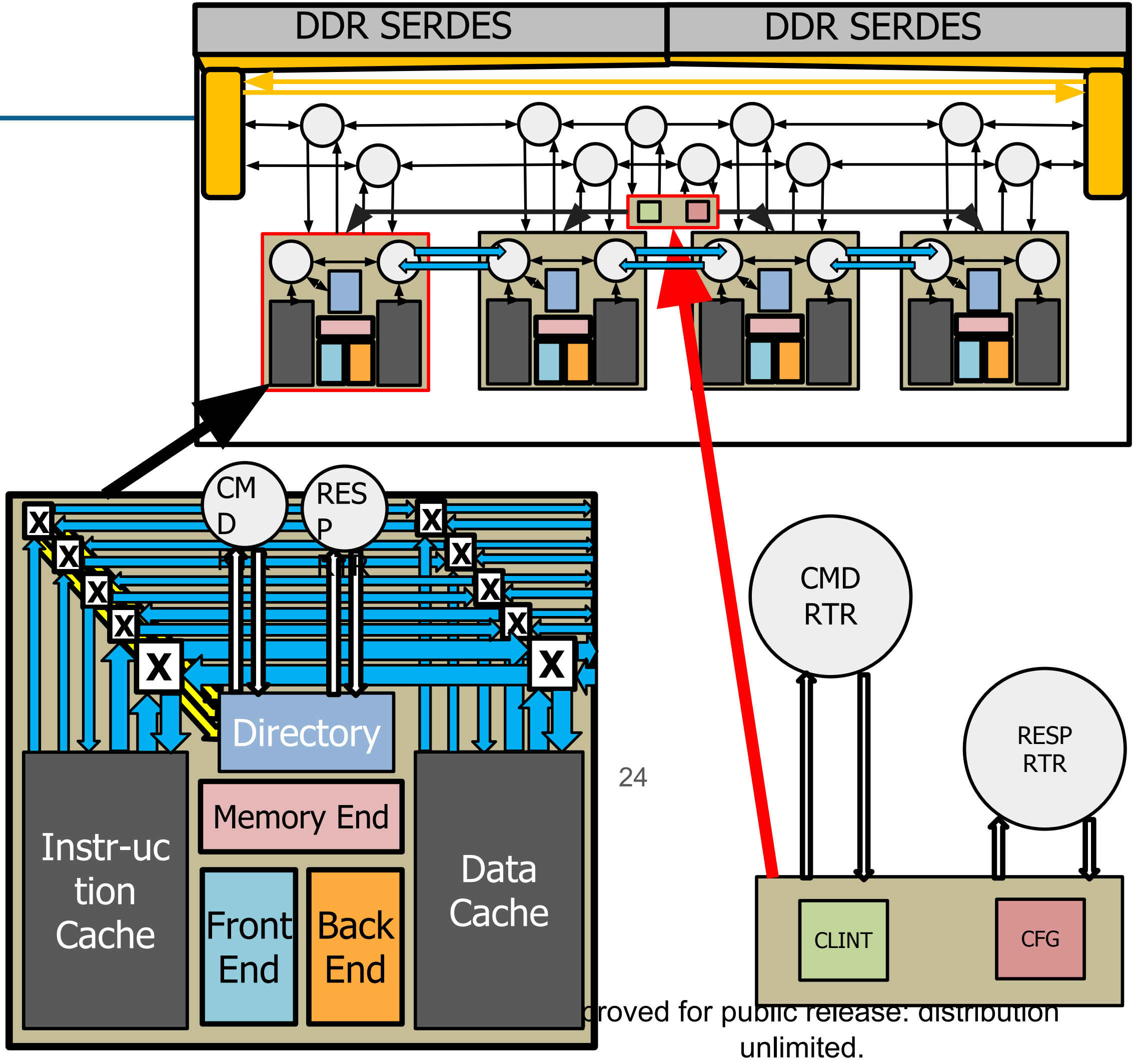
proved for public release: distribution unlimited.

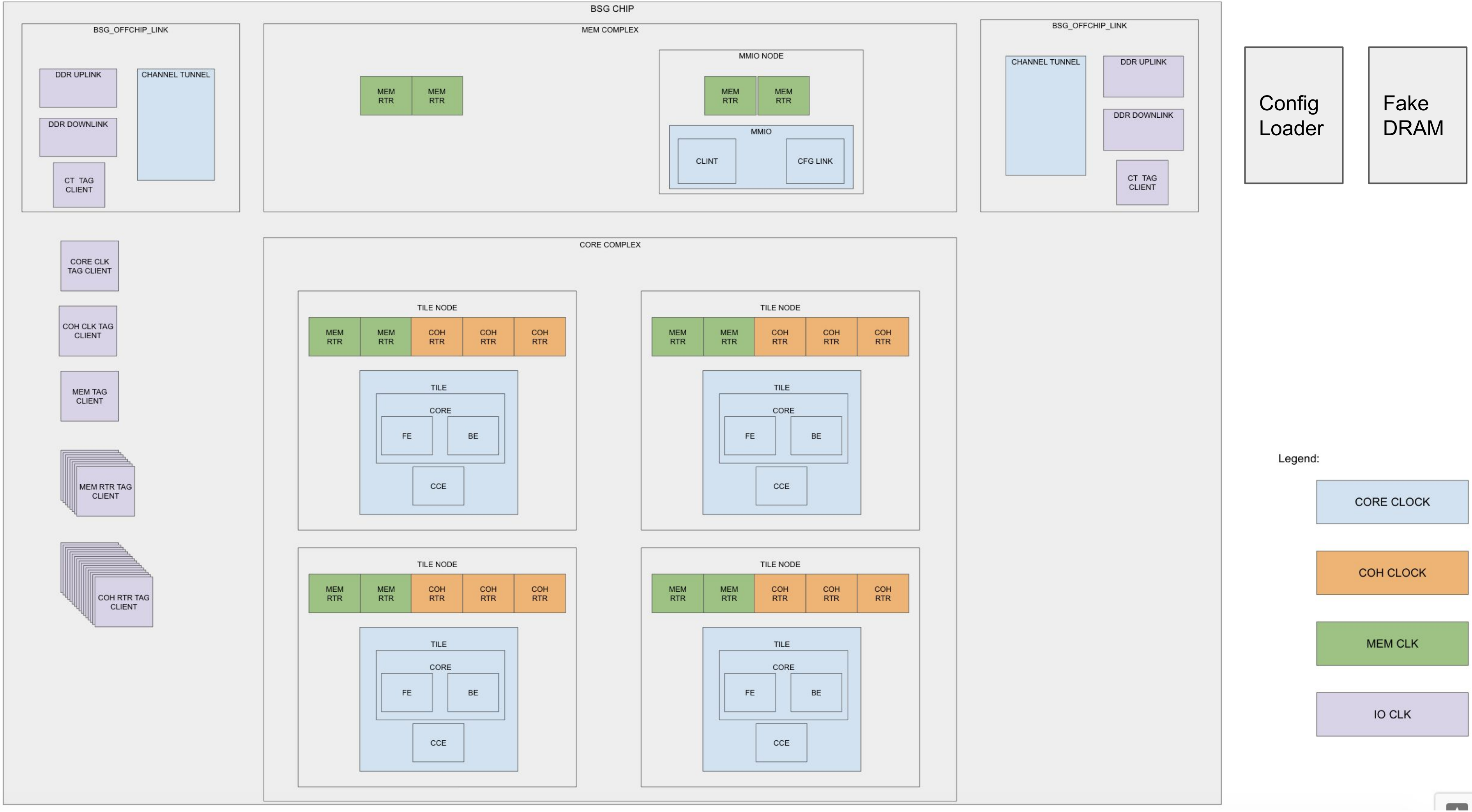
BlackParrot One ASIC

Taped out July 13, 2019!
GF 12nm Process Technology
3mm x 3mm die

4-core 64b RISC-V multicore
CLINT Interrupt Controller
Off-chip SERDES for DRAM and I/O
Chips can be chained

Each core:
- RV64IA with Virtual Memory
- Single-issue In-order
- 32K Data cache
- 32K Instruction cache
- 64-entry BTB
- 8-entry DTLB
- 8-entry ITLB





BSG CHIP

BSG_OFFCHIP_LINK

DDR UPLINK

DDR DOWNLINK

CT TAG CLIENT

CHANNEL TUNNEL

MEM COMPLEX

MEM RTR

MEM RTR

MMIO NODE

MEM RTR

MEM RTR

MMIO

CLINT

CFG LINK

BSG_OFFCHIP_LINK

CHANNEL TUNNEL

DDR UPLINK

DDR DOWNLINK

CT TAG CLIENT

Config Loader

Fake DRAM

CORE CLK TAG CLIENT

COH CLK TAG CLIENT

MEM TAG CLIENT

MEM RTR TAG CLIENT

COH RTR TAG CLIENT

CORE COMPLEX

TILE NODE

MEM RTR | MEM RTR | COH RTR | COH RTR | COH RTR

TILE

CORE

FE

BE

CCE

TILE NODE

MEM RTR | MEM RTR | COH RTR | COH RTR | COH RTR

TILE

CORE

FE

BE

CCE

TILE NODE

MEM RTR | MEM RTR | COH RTR | COH RTR | COH RTR

TILE

CORE

FE

BE

CCE

TILE NODE

MEM RTR | MEM RTR | COH RTR | COH RTR | COH RTR

TILE

CORE

FE

BE

CCE

Legend:

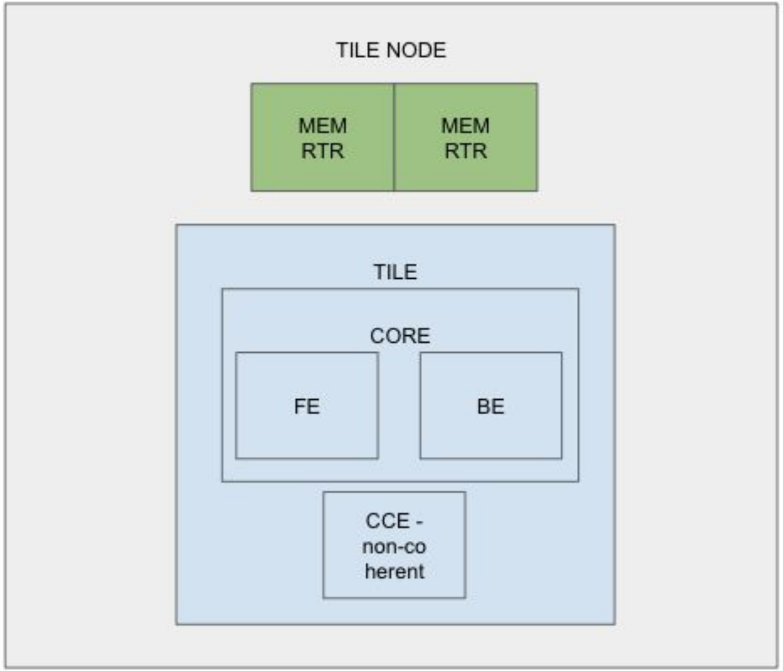
CORE CLOCK

COH CLOCK

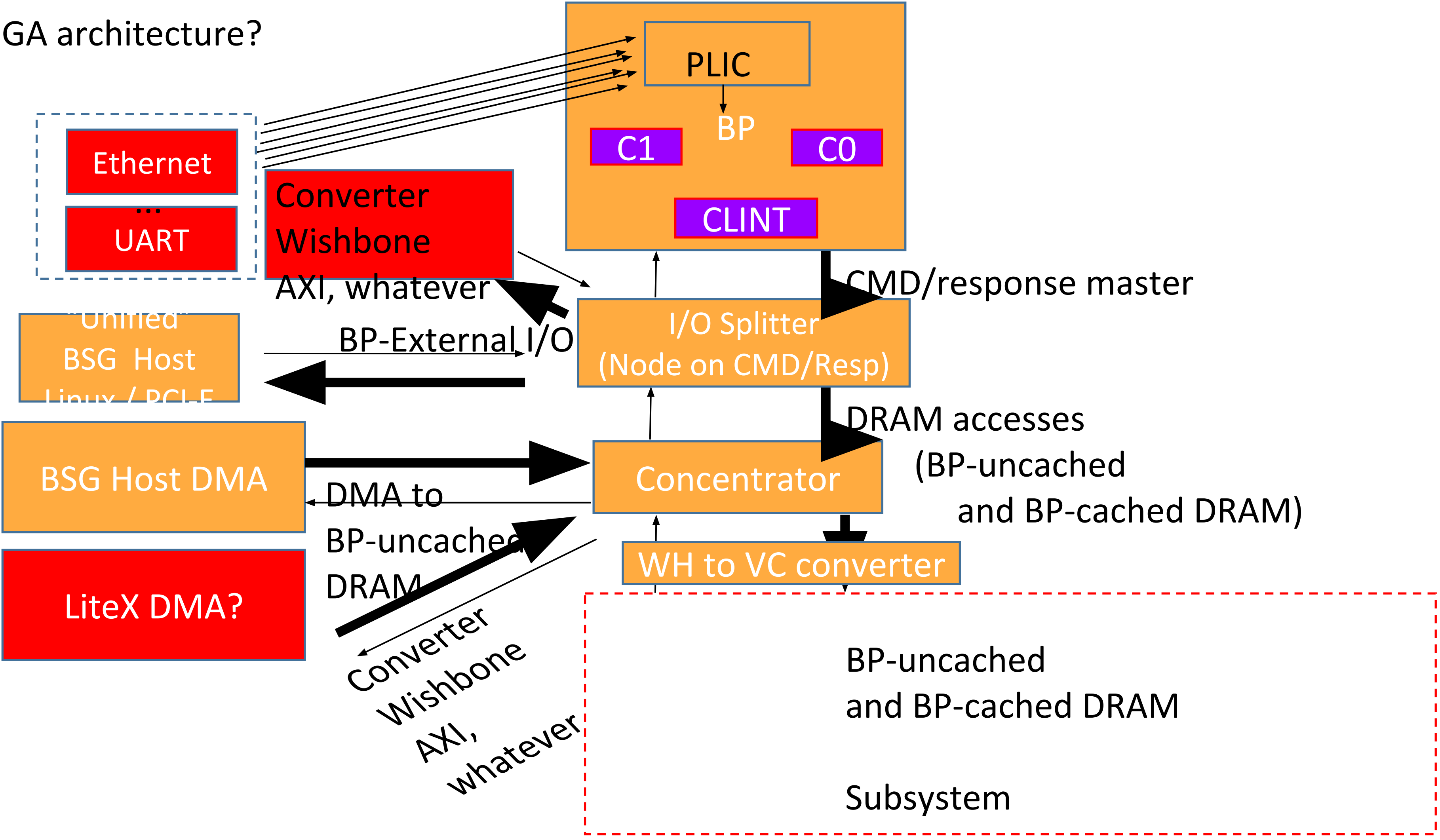
MEM CLK

IO CLK

Possible single core variant?



FPGA architecture?



PLIC

BP

C1

C0

CLINT

Ethernet

...

UART

Converter Wishbone AXI, whatever

CMD/response master

"Unified" BSG Host Linux / PCI-E

BP-External I/O

I/O Splitter (Node on CMD/Resp)

DRAM accesses (BP-uncached and BP-cached DRAM)

BSG Host DMA

Concentrator

DMA to BP-uncached DRAM

LiteX DMA?

WH to VC converter

Converter Wishbone AXI, whatever

BP-uncached and BP-cached DRAM

Subsystem

Software side

BlackParrot’s Growing Testsuite

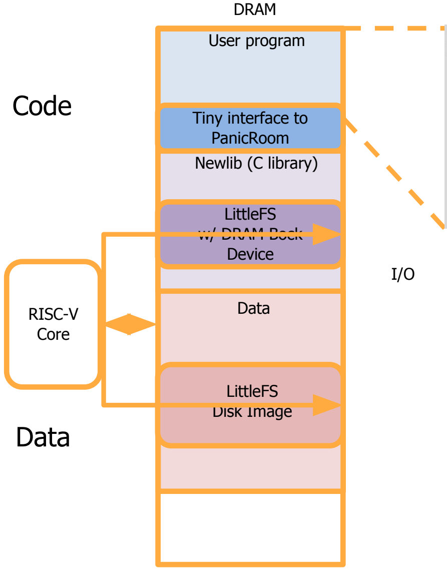
- **riscv-tests**: rv64ui-p-*/rv64ui-v-* - 121 unit tests, 7 benchmarks
  - https://github.com/riscv/riscv-tests
- **BEEBS**: Embedded benchmark suite - 77 tests
  - https://github.com/mageec/beebs
- **Coremark:** Industry-standard processor benchmark – 1 test
  - https://github.com/eembc/coremark
- **SPEC**: 1 test (VPR), more soon to come!
- **CMURPHI** – Formal verification of our cache coherence protocol
  - https://github.com/melver/cmurphi
- **AXE** – Runtime verification of memory consistency
  - https://github.com/CTSRD-CHERI/axe
- Include all tools and necessary patches in BlackParrot repo so that users can validate performance and functionality for themselves!

On Deck:

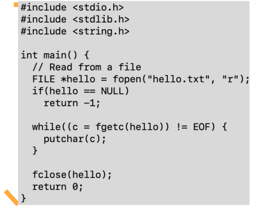
- Linux
- SQED
- riscv-dv
- csmith-based random testing
- riscv-formal

POSH: PanicRoom (Created for BlackParrot)

A C standard library that allows emerging agile architectures to rapidly run large test programs with POSIX I/O.
We compile the filesystem into the application!



PanicRoom: A C standard library with integrated file-system



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
  // Read from a file
  FILE *hello = fopen("hello.txt", "r");
  if(hello == NULL)
    return -1;

  while((c = fgetc(hello)) != EOF) {
    putchar(c);
  }

  fclose(hello);
  return 0;
}
```

Project	LOC
Rocket proxy kernel and fesvr tunnel & custom libgloss	14157
MIT RAW host-interface & custom libgloss	6999
UW HammerBlade using PanicRoom	**32**
UW BlackParrot using PanicRoom	**20**

A comparison of effort required
to support POSIX I/O
(in addition to newlib)

- We modified Newlib (an embedded C standard library) to sit on top of a tiny portable DRAM-based filesystem (ARM's open-source *LittleFS*) to support POSIX I/O on *bare-metal systems*.
- The file system disk image (with input files & stdin files) is compiled into the binary, and is read/write.
- Extremely portable: <40 lines of code required to port PanicRoom to a new architecture.
- Alternative approaches (like Berkeley Rocket, Ariane, and MIT Raw) proxy I/O to a host system and employ complex syscall translation facilities require reimplementation of RPC tunnels for each target: VCS, Verilator, emulation, ASIC, etc.
- >> 300 times less code

Goal: Release as open source as part of newlib shortly.

BlackParrot’s Growing Testsuite

- **riscv-tests**: rv64ui-p-*/rv64ui-v-* - 121 unit tests, 7 benchmarks
  - https://github.com/riscv/riscv-tests
- **BEEBS**: Embedded benchmark suite - 77 tests
  - https://github.com/mageec/beebs
- **Coremark:** Industry-standard processor benchmark – 1 test
  - https://github.com/eembc/coremark
- **SPEC**: 1 test (VPR), more soon to come!
- **CMURPHI** – Formal verification of our cache coherence protocol
  - https://github.com/melver/cmurphi
- **AXE** – Runtime verification of memory consistency
  - https://github.com/CTSRD-CHERI/axe
- Include all tools and necessary patches in BlackParrot repo so that users can validate performance and functionality for themselves!

On Deck:

- Linux
- SQED
- riscv-dv
- csmith-based random testing
- riscv-formal

Extra things to port

- Multi-core benchmarks
  - Parsec, SPLASH, etc.
  - Normally require an OS for thread management
    - Stub out pthreads? Could be incorporated in PanicRoom!
- riscv-dv
  - Automated UVM-based white-box assembly-driven tester
    - For compliance with RISC-V spec. Guaranteed to find bugs
- More SPEC benchmarks
  - Warning: long running. Let us know if you would like access, because SPEC is not open-source

More info, please

- Tracers for everything
  - TLB fills / evictions
  - Emulation logs
  - VM traces
  - Performance analysis tools (where did cycles go)
  - Coherence traffic packet widths/num flits/latencies
- Modeling of adding new instructions vs emulation
  - FPDIV SW vs simple hardware vs pipelined hardware
  - AMO in L1/L2/emulation

VLSI Side

FreePDK45

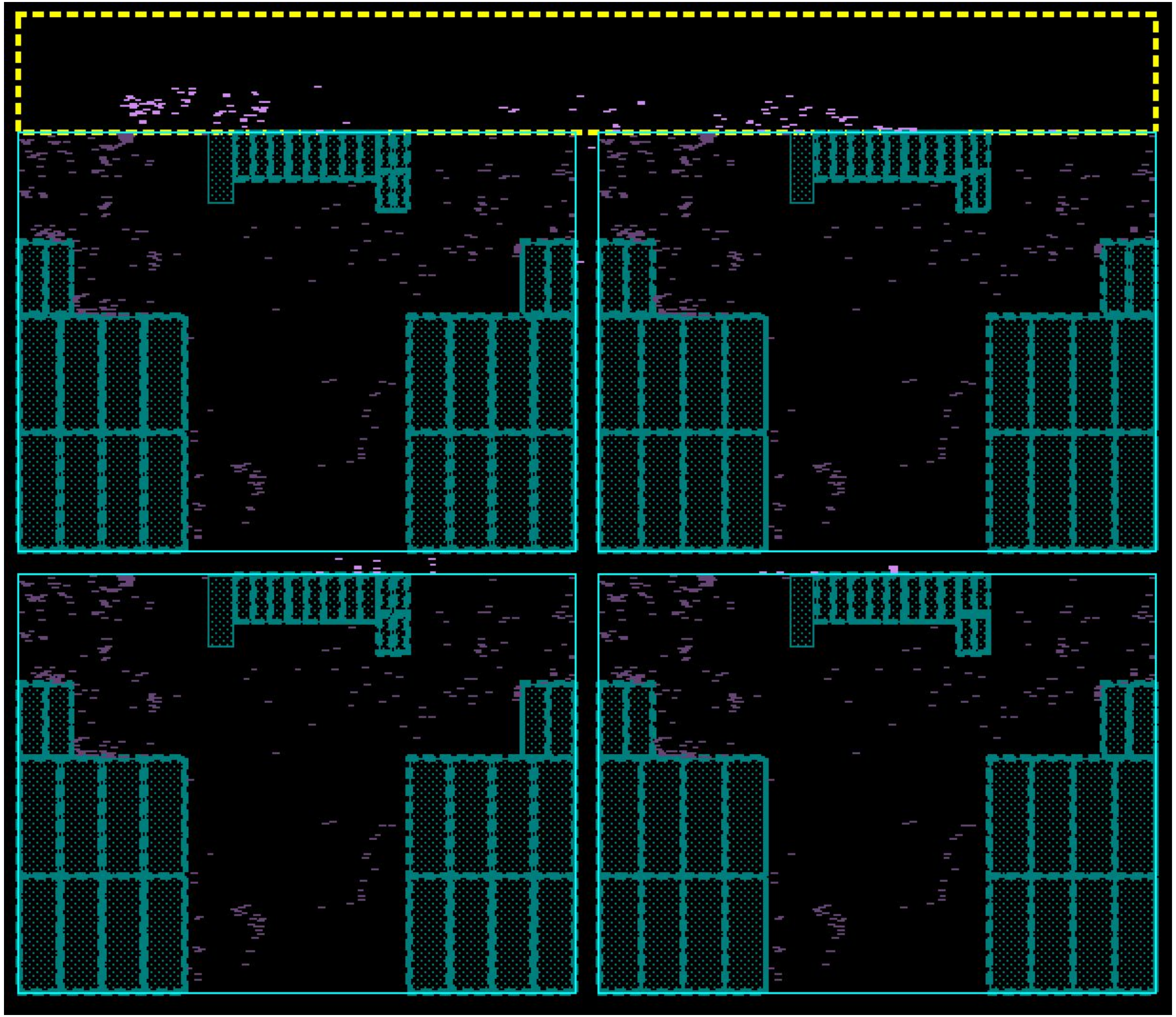
- Predictive 45nm modeling
- “Fake” PDK, but realistic enough to draw conclusions from

bsg_fakeram

- CACTI-based predictive SRAM generator
- Generates blackbox macros with .lib, .lef, .v

We have an ASIC flow set up!

- DC/ICC RM scripts are under NDA so we can’t quite publish it
- Contact me if you’d like your project to be in the VLSI space



Quick repo overview

- bp_common - Interface definitions and tool infrastructure
- bp_fe, bp_be, bp_me - End level modules
- bp_top - Top level (Core, SoC, FPGA wrappers)

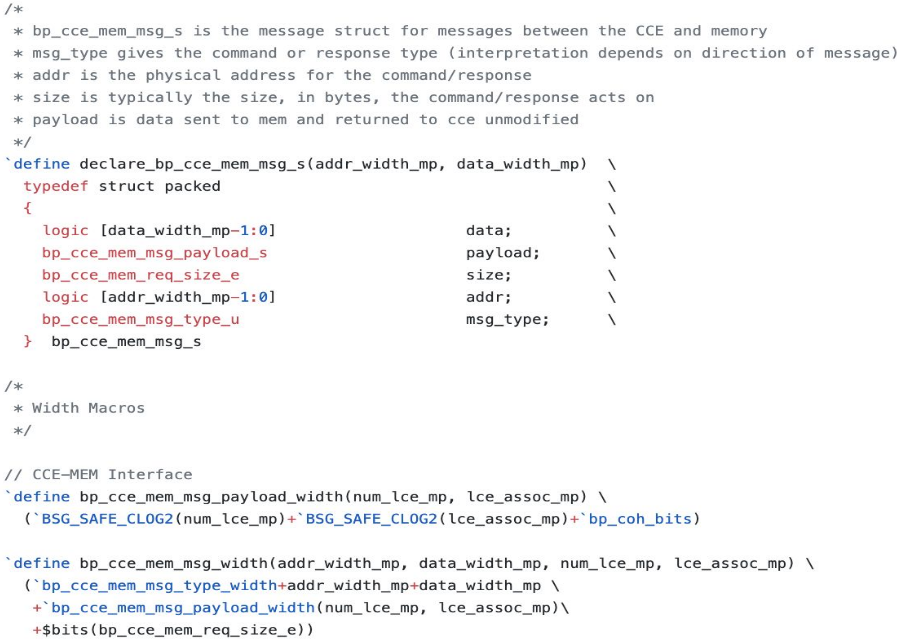
GETTING_STARTED.md on dev in the BlackParrot repo is the most up to date documentation

For now,

- All testbenches should be run from bp_top/syn
- All new tests should be added in bp_common/test

Parameterized structs

- SystemVerilog does not have a built in capability for parameterized structs
- We get around this by declaring macros which declare structs



```
/*
 * bp_cce_mem_msg_s is the message struct for messages between the CCE and memory
 * msg_type gives the command or response type (interpretation depends on direction of message)
 * addr is the physical address for the command/response
 * size is typically the size, in bytes, the command/response acts on
 * payload is data sent to mem and returned to cce unmodified
 */
`define declare_bp_cce_mem_msg_s(addr_width_mp, data_width_mp)  \
  typedef struct packed                                         \
  {                                                             \
    logic [data_width_mp-1:0]                    data;          \
    bp_cce_mem_msg_payload_s                     payload;       \
    bp_cce_mem_req_size_e                        size;          \
    logic [addr_width_mp-1:0]                    addr;          \
    bp_cce_mem_msg_type_u                        msg_type;      \
  }  bp_cce_mem_msg_s

/*
 * Width Macros
 */

// CCE-MEM Interface
`define bp_cce_mem_msg_payload_width(num_lce_mp, lce_assoc_mp) \
  (`BSG_SAFE_CLOG2(num_lce_mp)+`BSG_SAFE_CLOG2(lce_assoc_mp)+`bp_coh_bits)

`define bp_cce_mem_msg_width(addr_width_mp, data_width_mp, num_lce_mp, lce_assoc_mp) \
  (`bp_cce_mem_msg_type_width+addr_width_mp+data_width_mp \
   +`bp_cce_mem_msg_payload_width(num_lce_mp, lce_assoc_mp)\
   +$bits(bp_cce_mem_req_size_e))
```

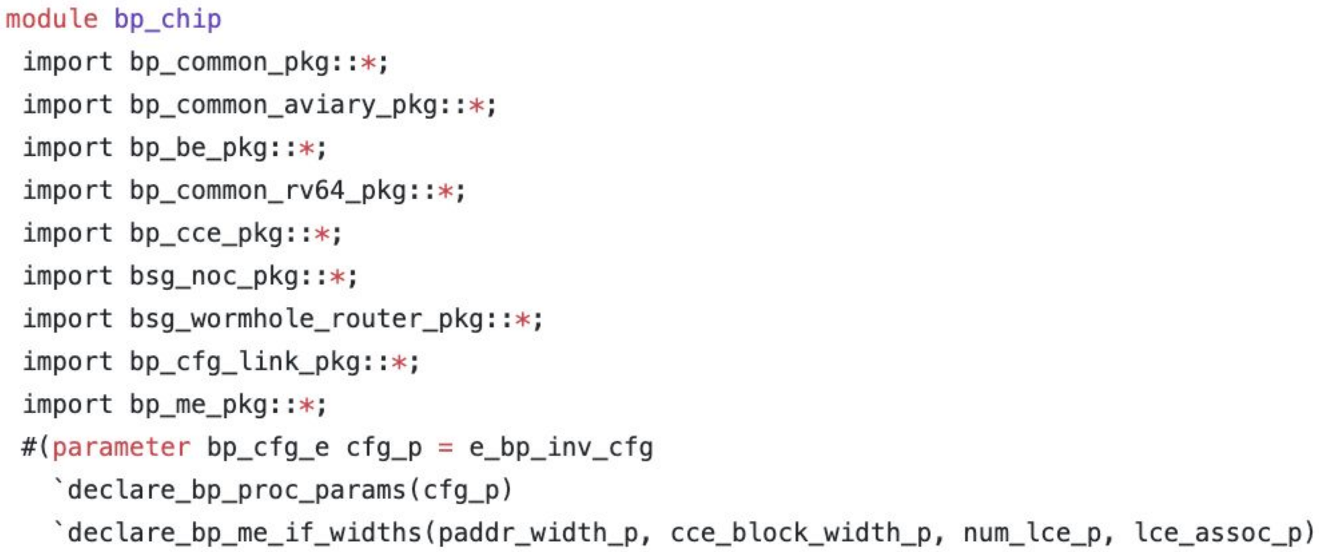
- Why not use $bits for port widths?
  - Structs are declared inside of modules, because that’s where parameters are scoped. Therefore the struct does not exist at port elaboration time!
- Why not declare parameters globally and have structs declared once?
  - More flexibility, could have big.LITTLEParrot
  - Keeps modules more generic, no dependency on higher level parameters

How to customize BP

- To get a handle on the knobs that BlackParrot has, we use a struct of parameters to declare all high level parameterized structs
- You can find validated parameter sets in bp_common_aviary_pkg.vh
- https://github.com/black-parrot/pre-alpha-release/blob/master/bp_common/src/include/bp_common_aviary_pkg.vh
- To add a new parameter to the set, add to bp_common_aviary_defines.vh

How to customize BP

- In the code to gain back all of the toplevel parameters, simply use `declare_proc_params(cfg_p)



```
module bp_chip
 import bp_common_pkg::*;
 import bp_common_aviary_pkg::*;
 import bp_be_pkg::*;
 import bp_common_rv64_pkg::*;
 import bp_cce_pkg::*;
 import bsg_noc_pkg::*;
 import bsg_wormhole_router_pkg::*;
 import bp_cfg_link_pkg::*;
 import bp_me_pkg::*;
 #(parameter bp_cfg_e cfg_p = e_bp_inv_cfg
   `declare_bp_proc_params(cfg_p)
   `declare_bp_me_if_widths(paddr_width_p, cce_block_width_p, num_lce_p, lce_assoc_p)
```

BlackParrot: Community driven uarch

- BlackParrot is a relatively new project, rough edges and all
- We’re also bootstrapping a ton of open-source infrastructure - SW, HW, system-level, ASIC design
- The best way to help the project is to raise issues where things are unclear or incorrect, especially in HOWTO guides
- The easiest way to get commits into the main BlackParrot repo is submitting documentation patches
- Your project can help us pathfind new features, add visibility and make BlackParrot the best default option for computer architecture research!