SIGIL

Classifying Workload Communication

Mike Lui
PhD student - Drexel University

Dr. Siddharth Nilakantan
Nvidia
Graduated - Drexel University

Dr. Baris Taskin
Associate Professor - Drexel University

Dr. Mark Hempstead
Associate Professor - Tufts University

Sigil Release

- Official Website
  - http://dpac.ece.drexel.edu/current-research-projects/sigil/
- Contact: michael.d.lui@drexel.edu
- Related Publications
  - "Platform-independent Analysis of Function-level Communication in Workloads", Siddharth Nilakantan and Mark Hempstead, IISWC 2013
  - "Metrics for Early-Stage Modeling of Many-Accelerator Architectures", Siddharth Nilakantan, Steven Battle and Mark Hempstead, CAL July-Dec 2012

Getting Sigil

- Available open source
  - **git clone** https://github.com/snilakan/Sigil
  - Documentation included

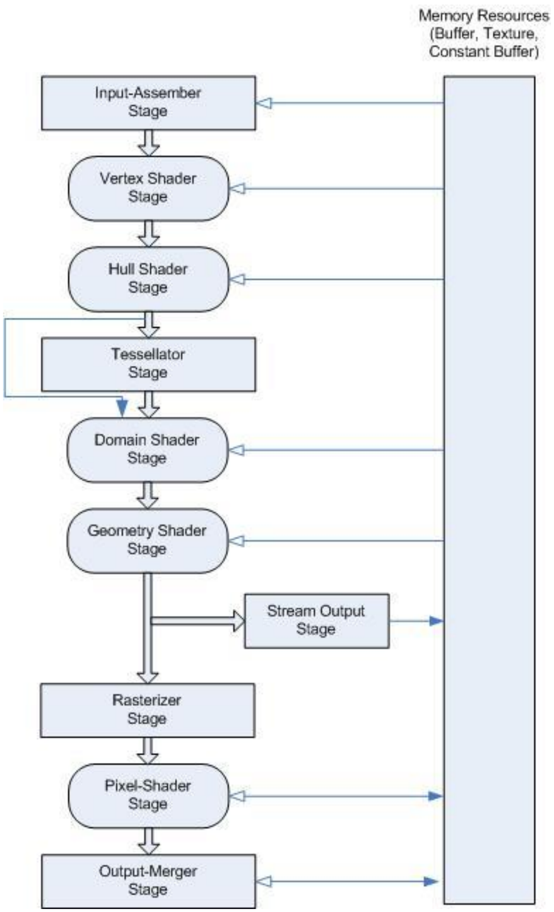
- Tested and validated in Linux
  - Officially tested distros: CentOS**6**, Ubuntu **12.04** LTS, Ubuntu **14.04** LTS
  - Supported by any system supported by Valgrind (**3.10.1**)

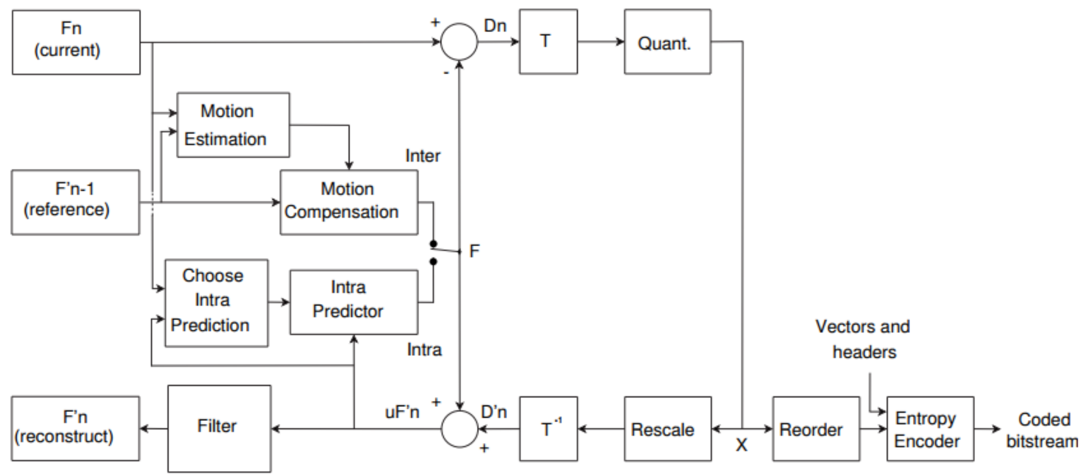
Outline

- **Accelerator Selection Problem**
  - Example
- Sigil Overview
- Sigil Methodology for Accelerator Selection
- Partitioning Example
- Building and Running Sigil

Motivational Applications

- Pipelined parallel apps typically chosen for HW acceleration



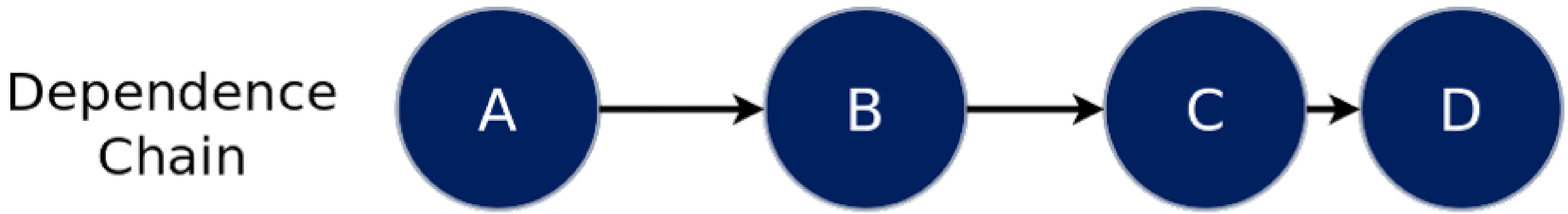


What is accelerator selection?

- Which functions to accelerate?

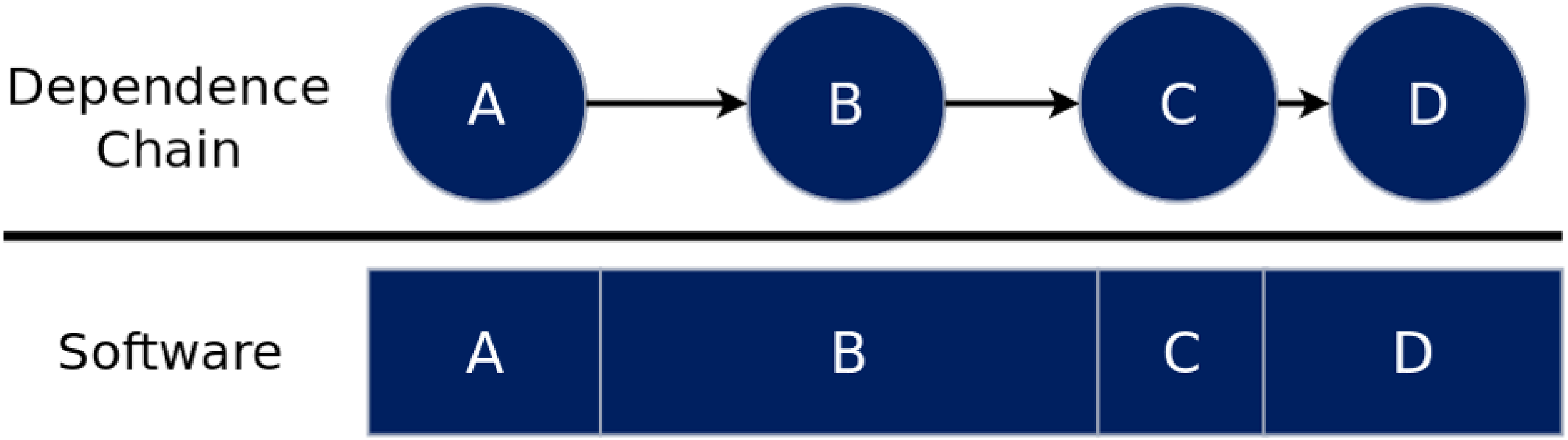
What is accelerator selection?

- Which functions to accelerate?
  - What are limiting factors for selection?



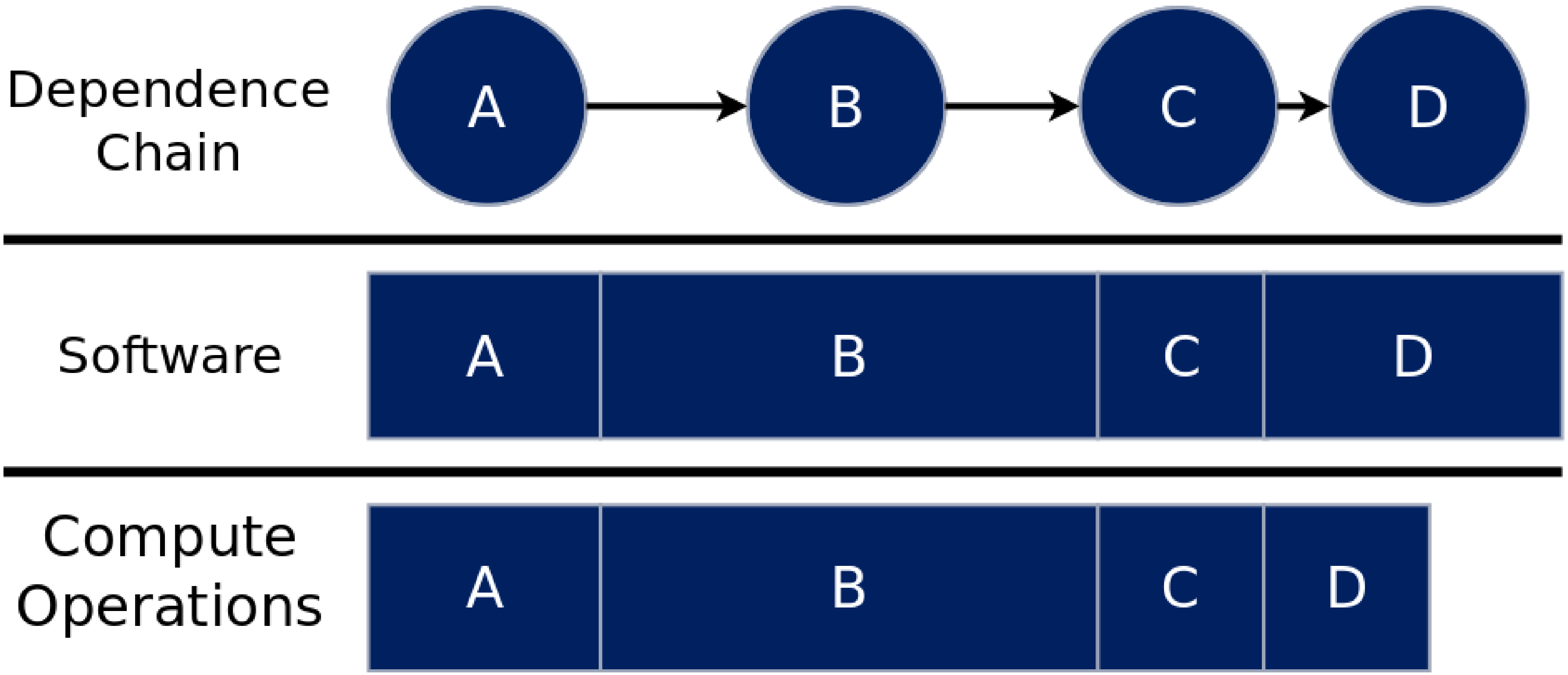
What is accelerator selection?

- Which functions to accelerate?
  - What are limiting factors for selection?

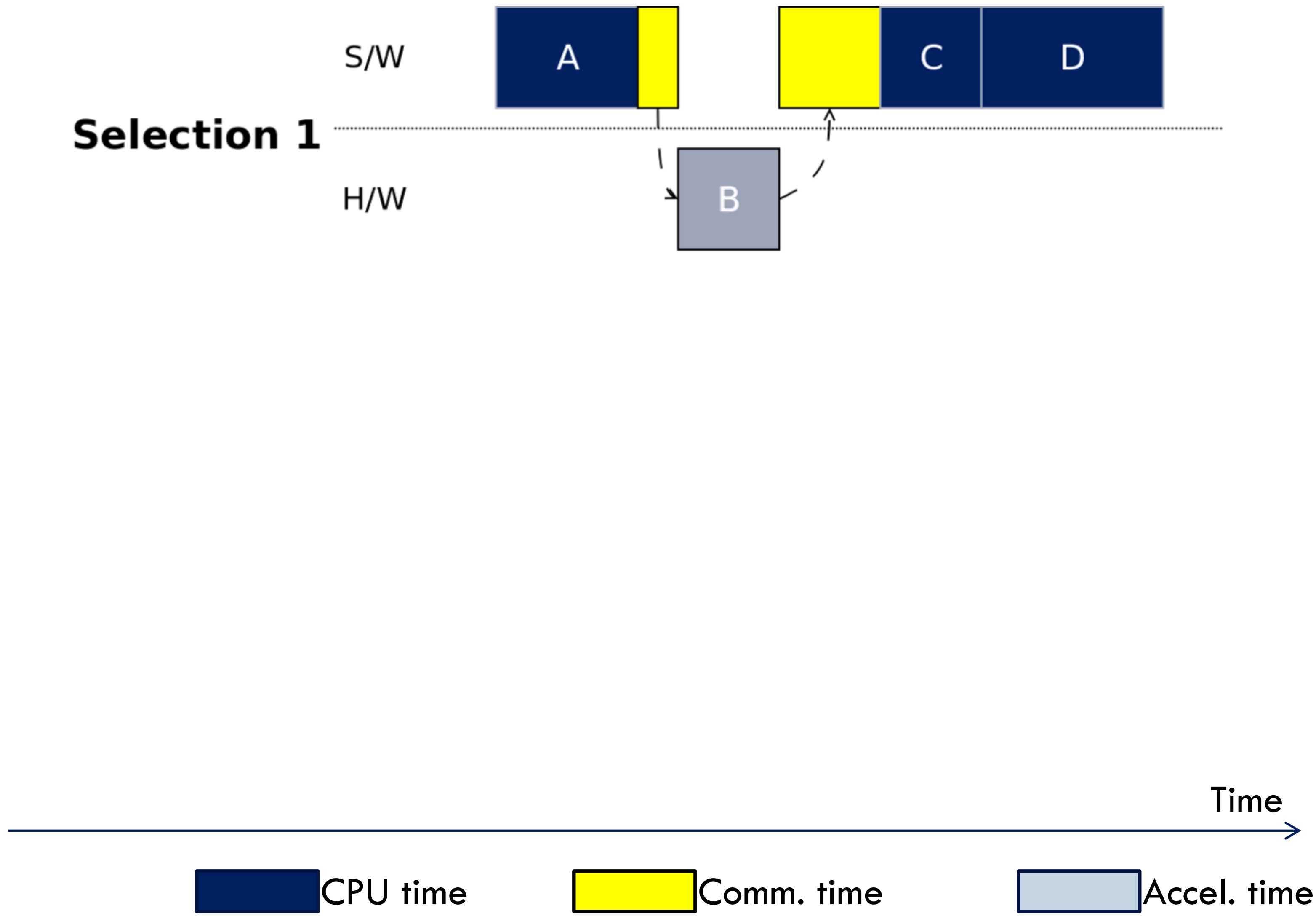


What is accelerator selection?

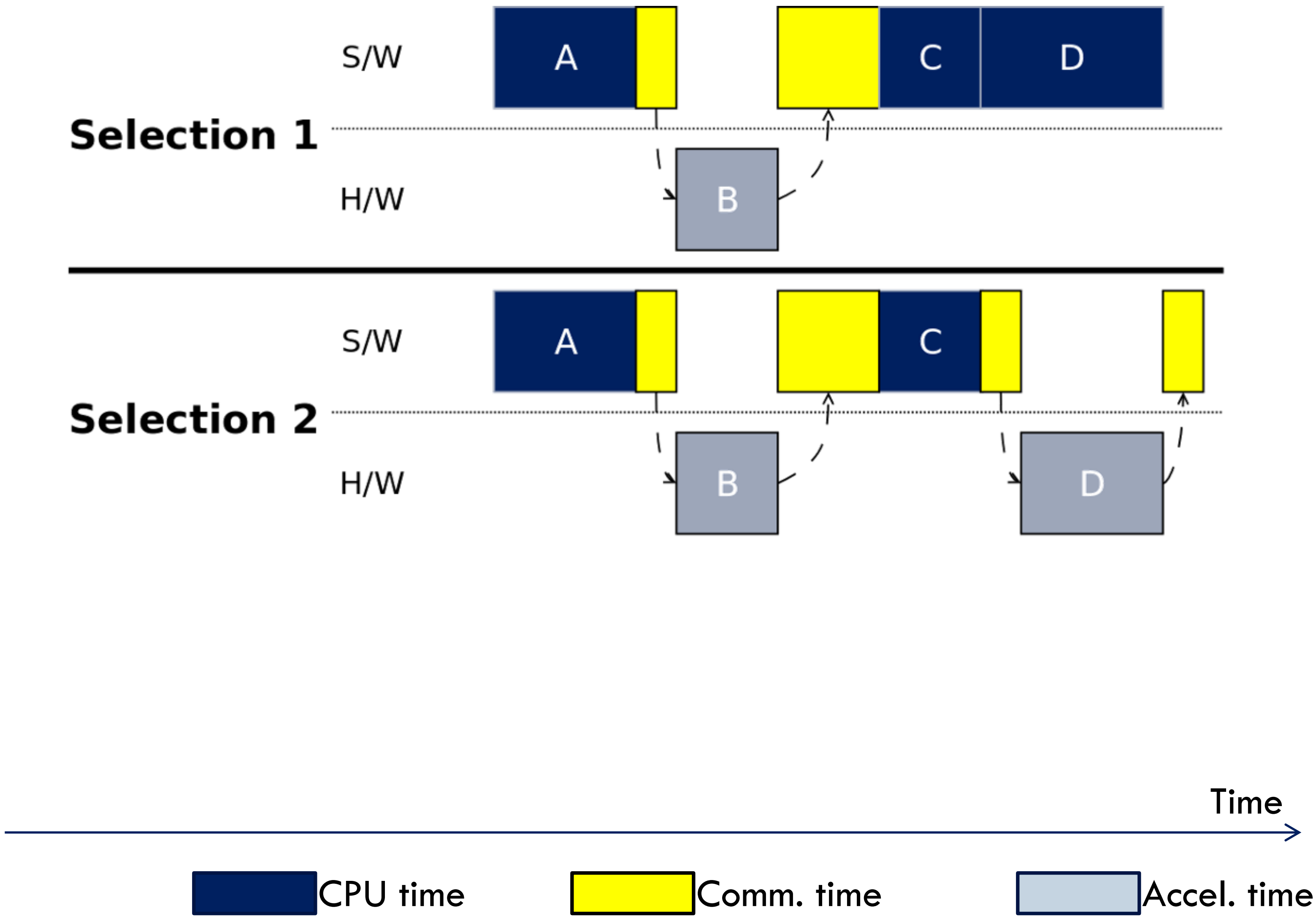
- Which functions to accelerate?
  - What are limiting factors for selection?



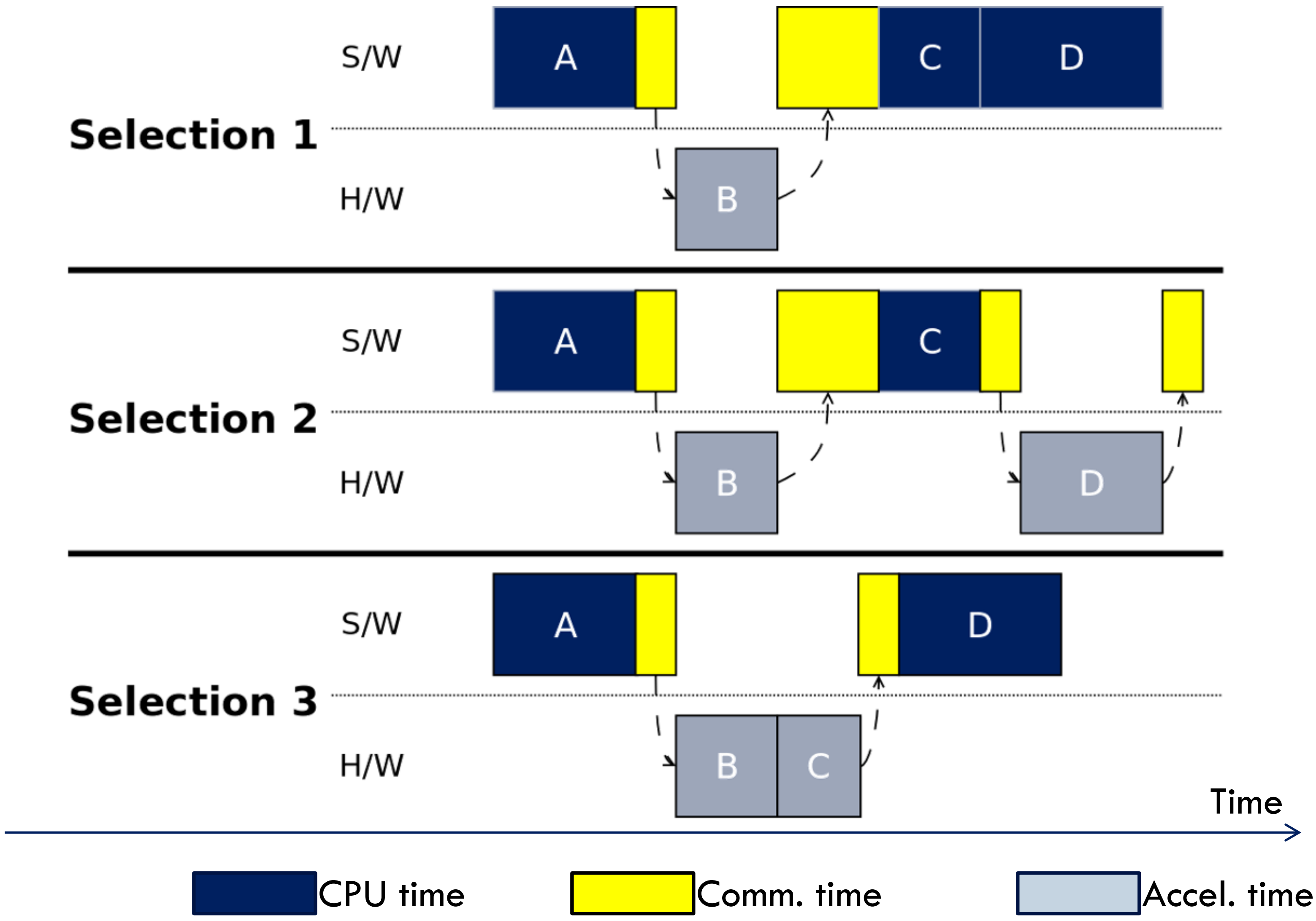
Accelerator selection



Accelerator selection



Accelerator selection



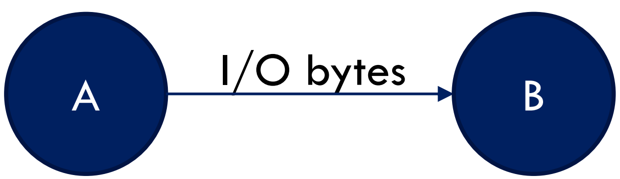
Platform-independent metrics

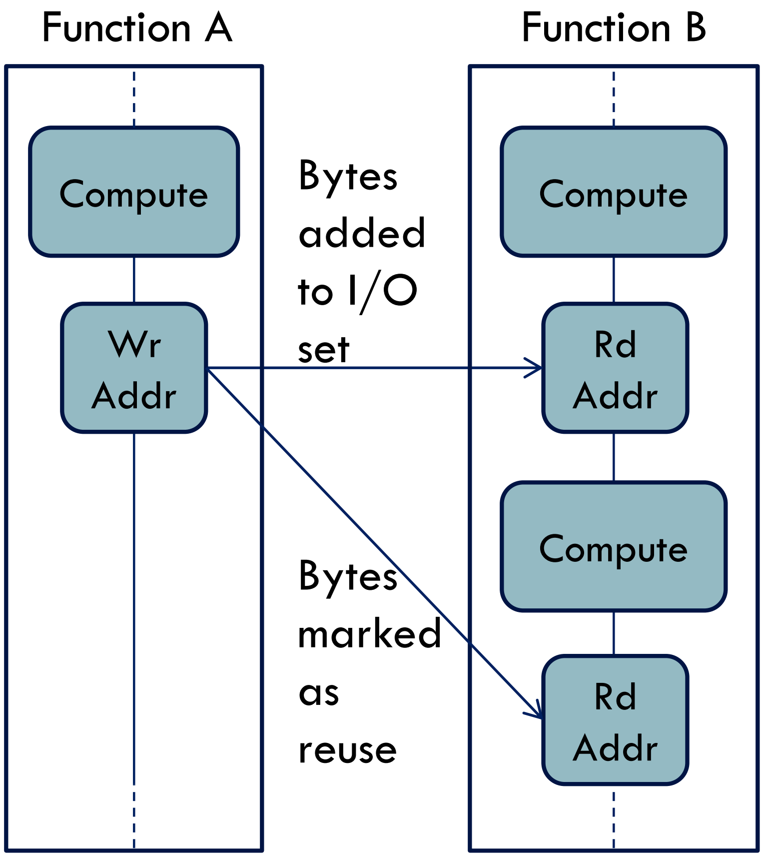
- Accelerator time and communication time are implementation-dependent!
  - Large design space for implementations

- Early stage design approach: Capture platform-independent metrics as proxy
  - Accelerator time → Compute operations
  - Communication Time → I/O set of bytes for each function

Capturing Input/Output Set

- Input/Output set: **NOT all** memory reads and writes, only unique ones
- Biggest challenge: Measuring *unique* communication





Enter Sigil

- Novelty: **Sigil** measures these metrics **automatically**
  - Classifying communication (unique and total bytes)
  - Compute operations for each function
  - Produces control data flow graph (CDFG) representations

- Revisit the Q: Which functions to accelerate?
  - Apply HW/SW partitioning algorithm to graphs!
  - Goals of algorithm
    - Minimize *unique* communication
    - Maximizing coverage in HW

Outline

- Accelerator selection problem
- **Sigil Overview**
- Sigil Methodology for Accelerator Selection
- Partitioning Example
- Building and Running Sigil

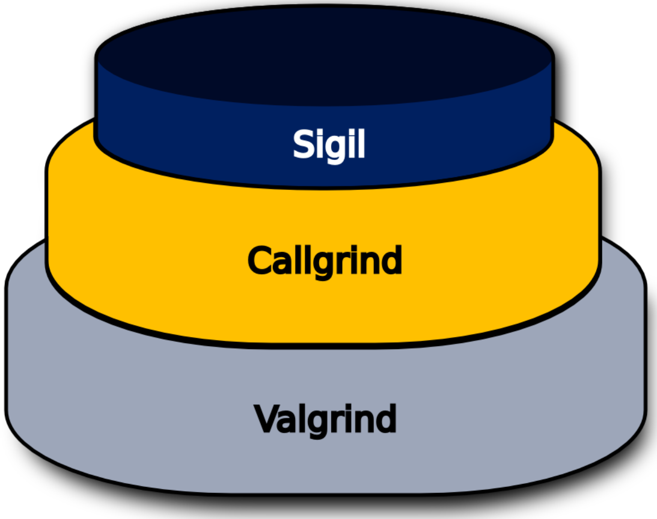
Sigil Implementation

- Implemented into Callgrind
- Works on binary, no source changes
- Can be implemented on any framework. Requires
  - Functions
  - Load/Store addresses

- Control data flow graph
- Unique and local communication costs and edges

- Cache simulation
- Branch prediction

- Dynamic binary instrumentation
- VEX IR generation

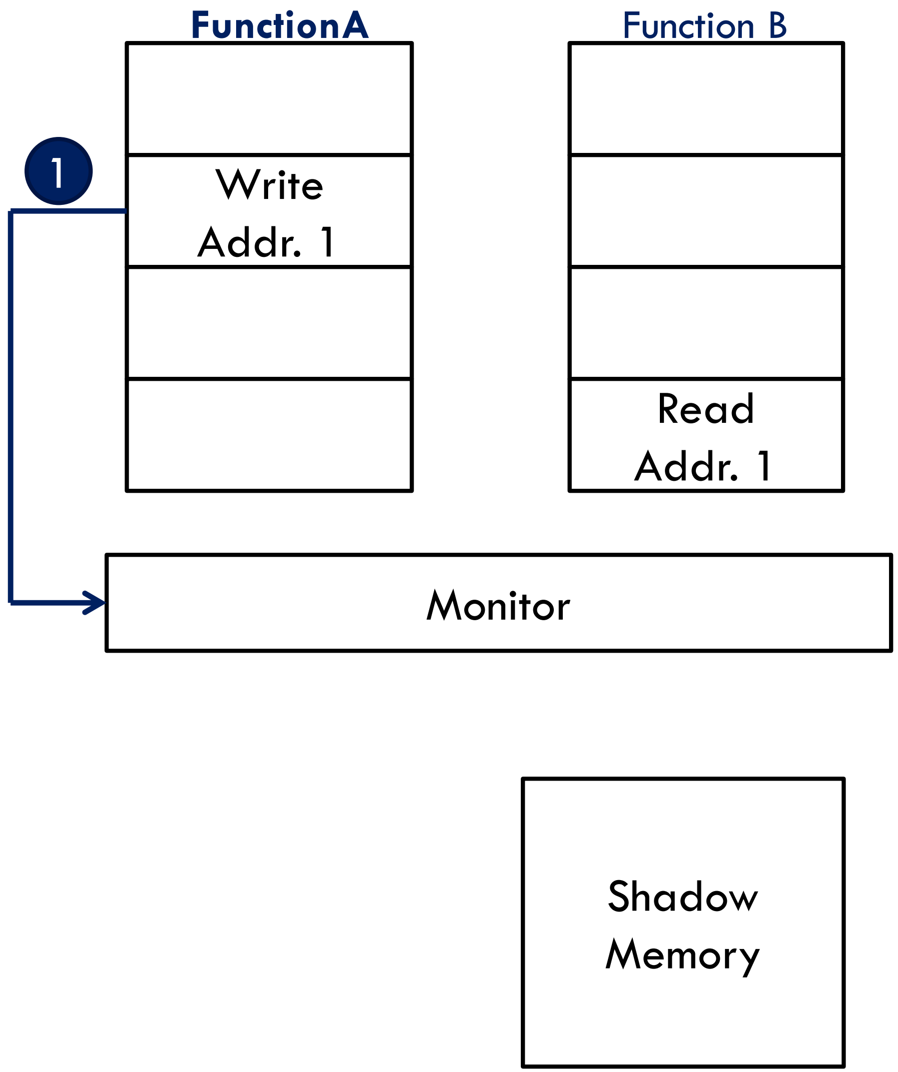


Sigil - Binary Instrumentation

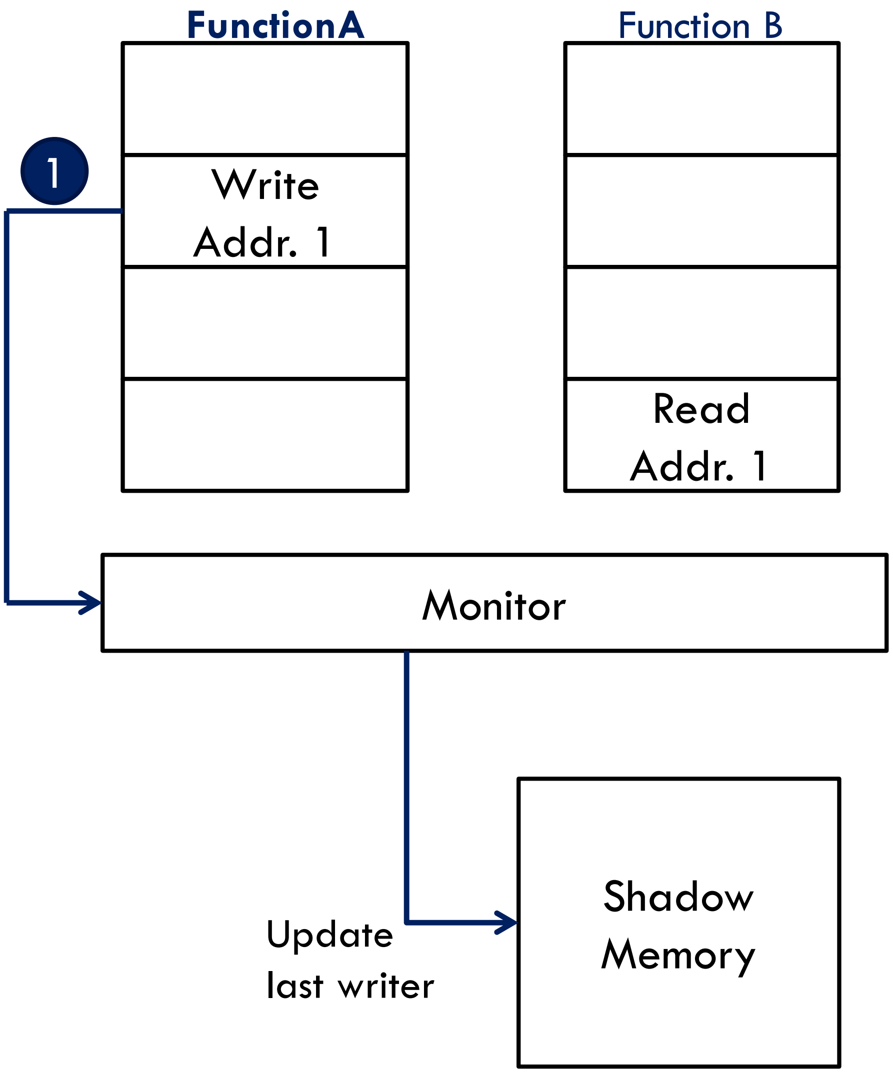
- Why Valgrind?
  - VEX IR provides
    - Abstract compute
    - Abstract load/store
    - Inspect every byte
    - Memory addresses and widths
  - Callgrind provides
    - function calls, returns, et al
  - Multi-platform support
    - Mature Linux support

Tracking unique communication

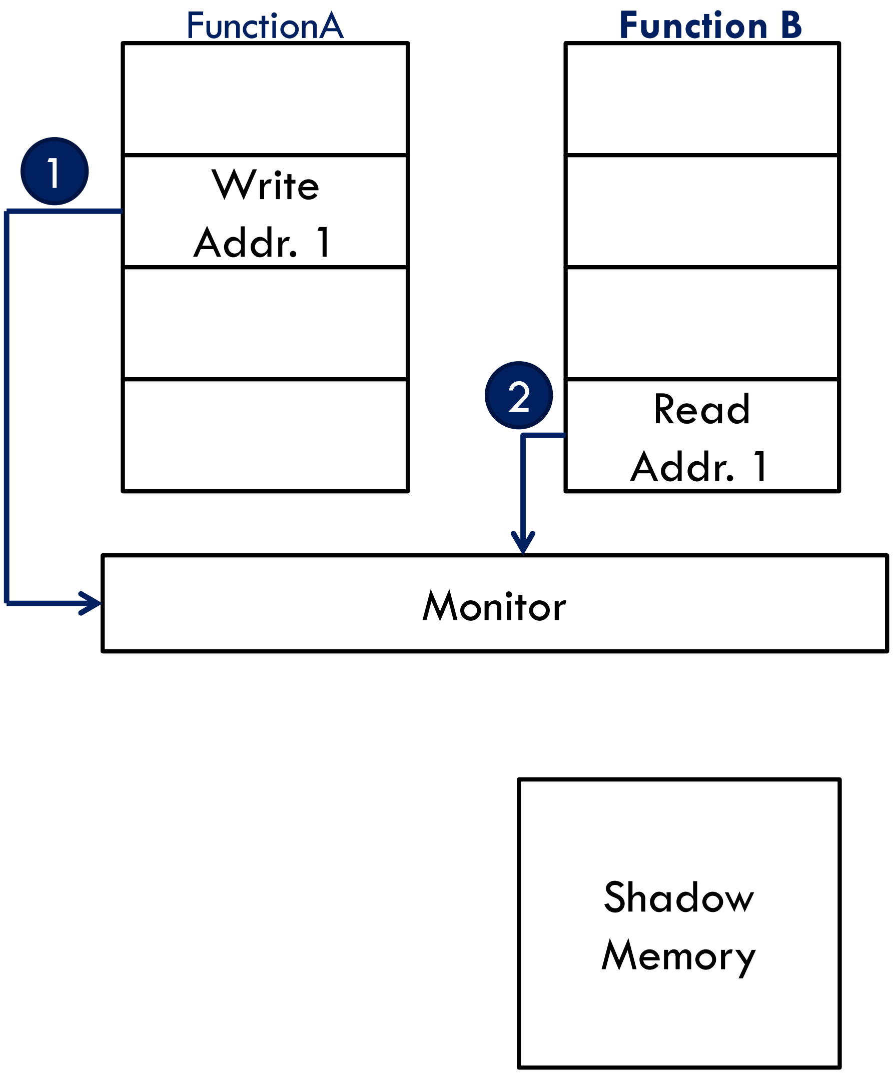
Tracking unique communication



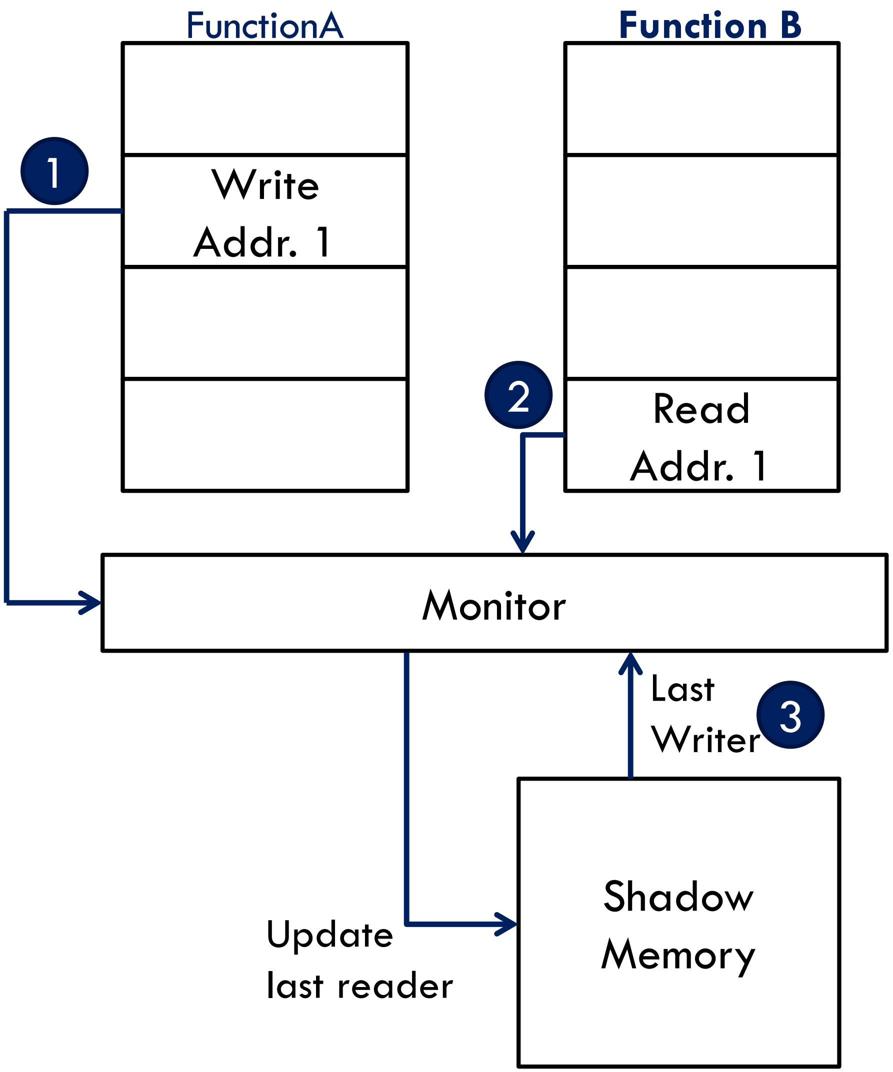
Tracking unique communication



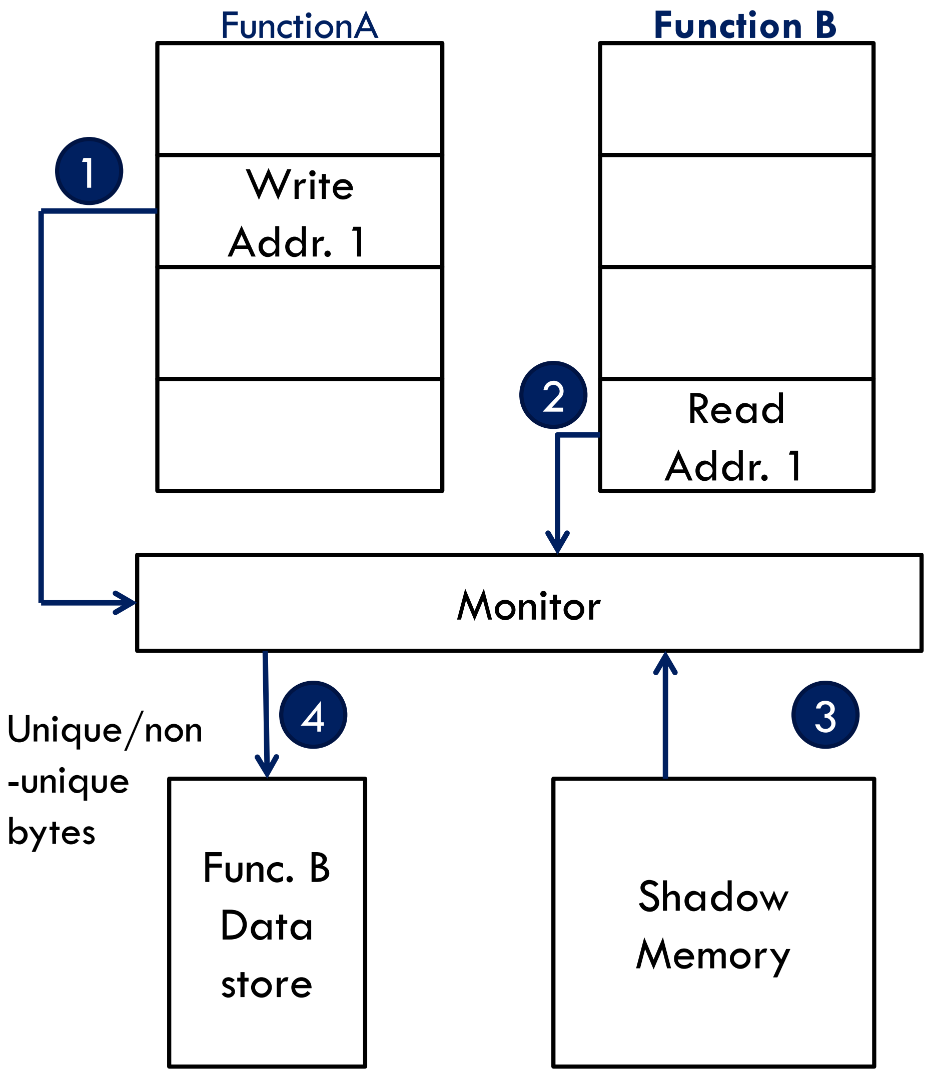
Tracking unique communication



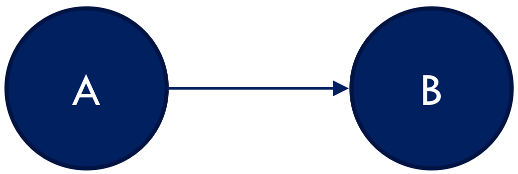
Tracking unique communication



Tracking unique communication



Inside Shadow Memory

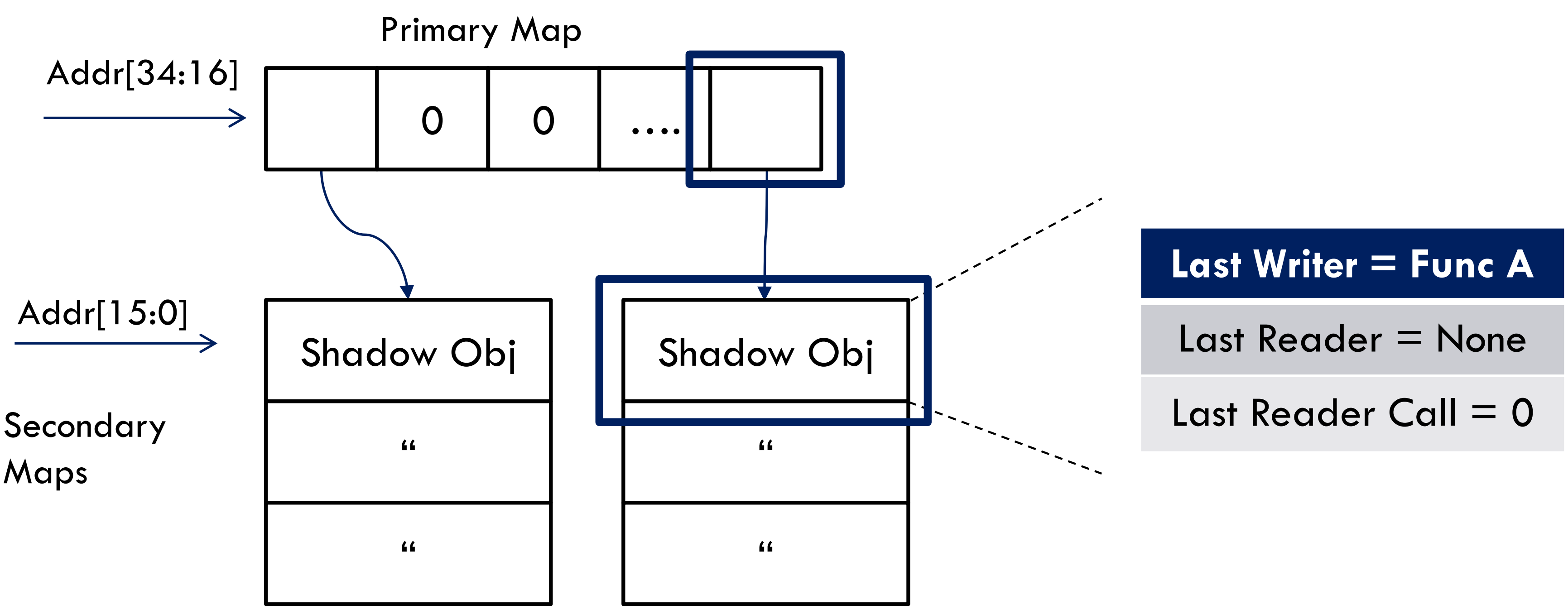


ST Addr, Register in Function A

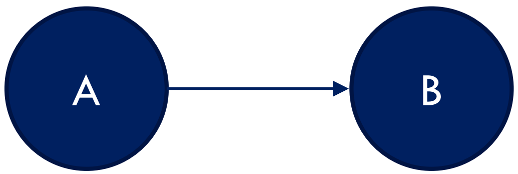
.

.

LD Register, Addr in Function B



Inside Shadow Memory

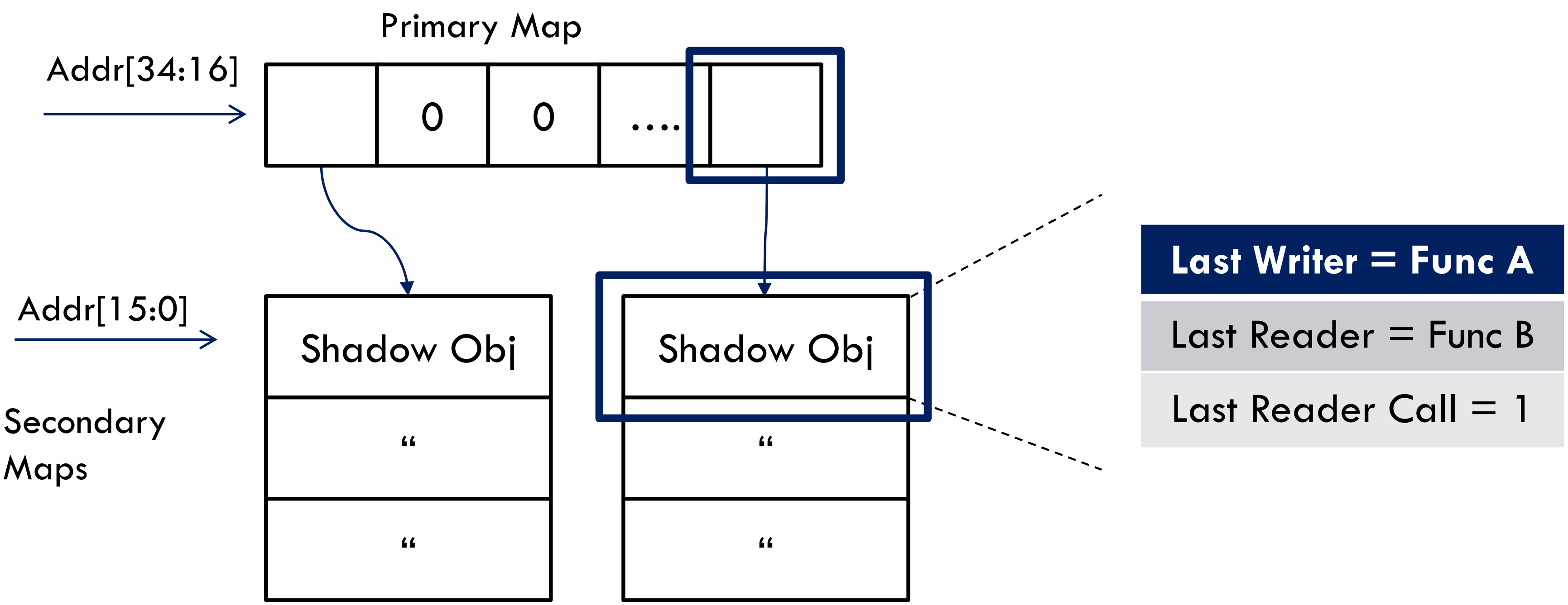


ST Addr, Register in Function A

.

.

LD Register, Addr in Function B

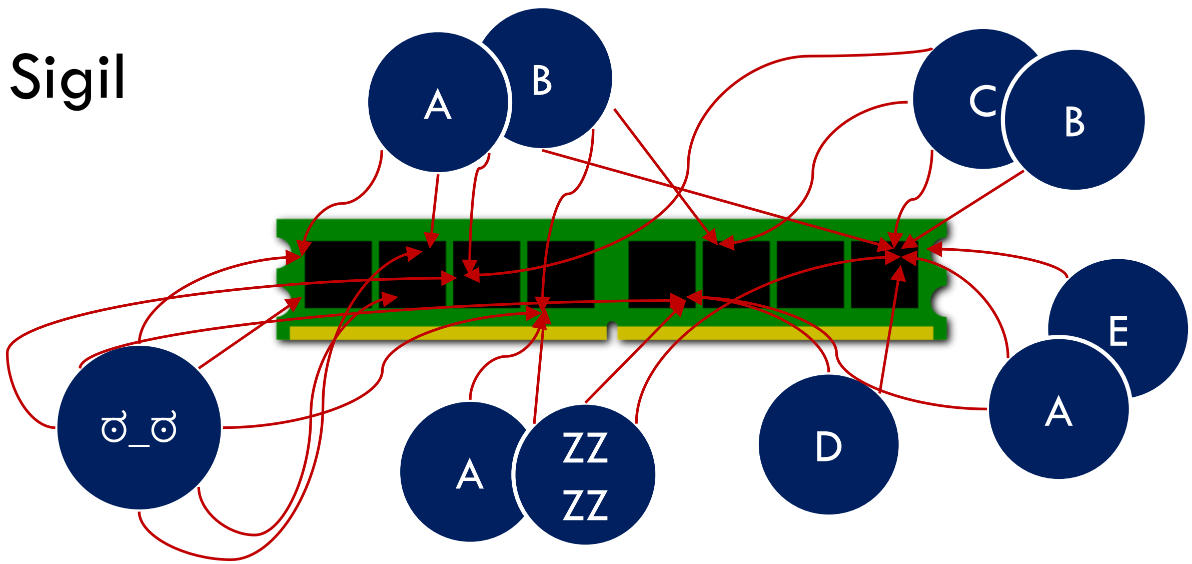


Inside Shadow Memory

- Challenges & Considerations
  - Redesigning shadow memory
    - Need to track more state than memcheck
  - Memcheck
    - 1-bit addressable
    - 1-bit valid
    - 1-void* LIVE heap locations
    - Some alignment state
    - Heuristic algorithms developed over time

Inside Shadow Memory

- Challenges & Considerations
  - Redesigning shadow memory
    - Need to track more state than memcheck
  - Sigil



Inside Shadow Memory

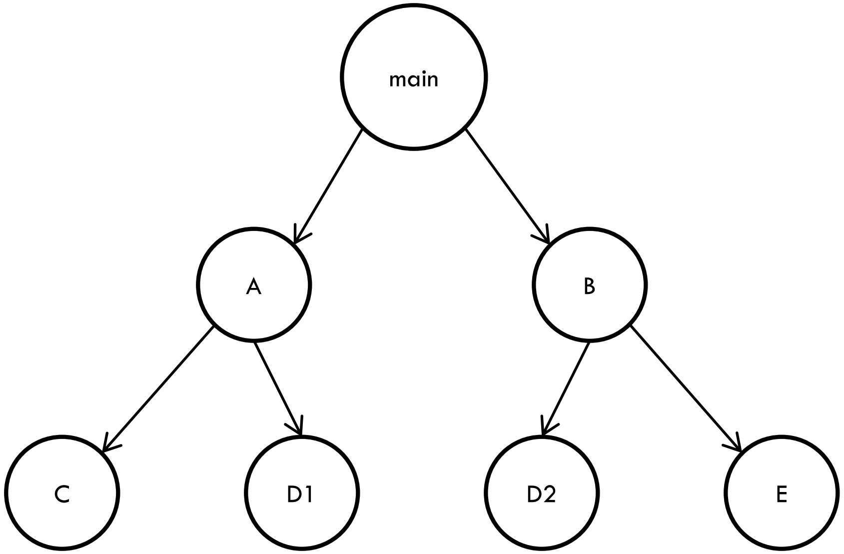
- Challenges & Considerations
  - Redesigning shadow memory
    - Need to track more state than memcheck
  - Sigil resources
    - – 2GB of user space memory → 34GB minimum!
    - +Only need to run once

Outline

- Accelerator selection problem
- Sigil Overview
- **Sigil Methodology for Accelerator Selection**
  - Control Data flow graphs
  - Partitioning process
- Partitioning Example
- Building and Running Sigil

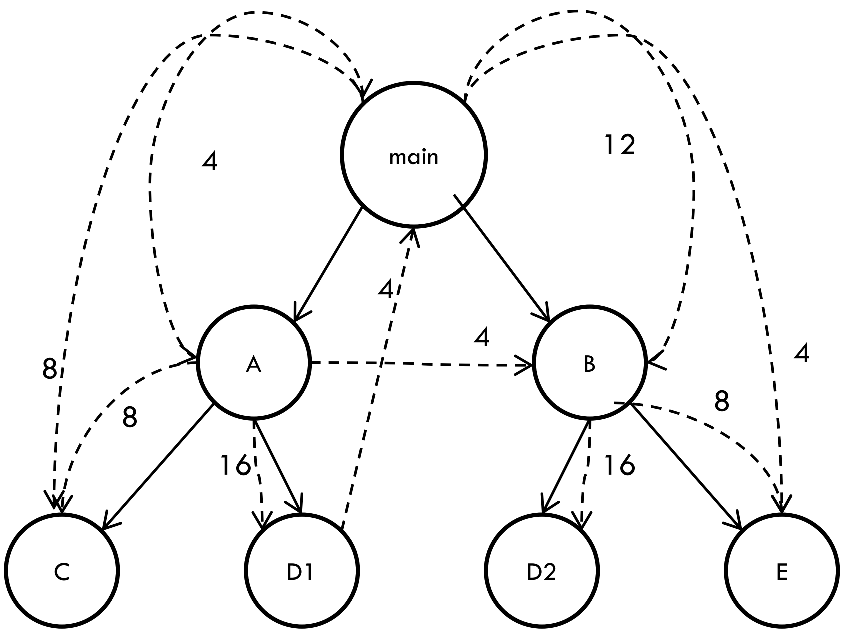
Control Data Flow graphs

- Function calltrees….
  - Hierarchical representation of functions in application
  - Obtained via Callgrind



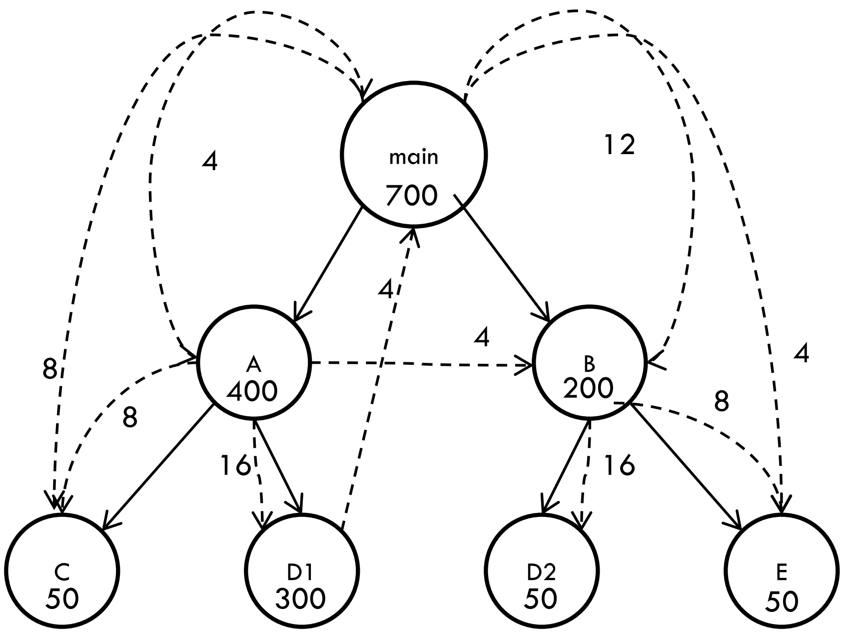
Control Data Flow graphs

- Function calltrees annotated with *unique* communication flow
  - Obtained via Sigil



Control Data Flow graphs

- Function calltrees annotated with *unique* communication flow
  - Add computation costs in as well
  - Also obtained via Sigil

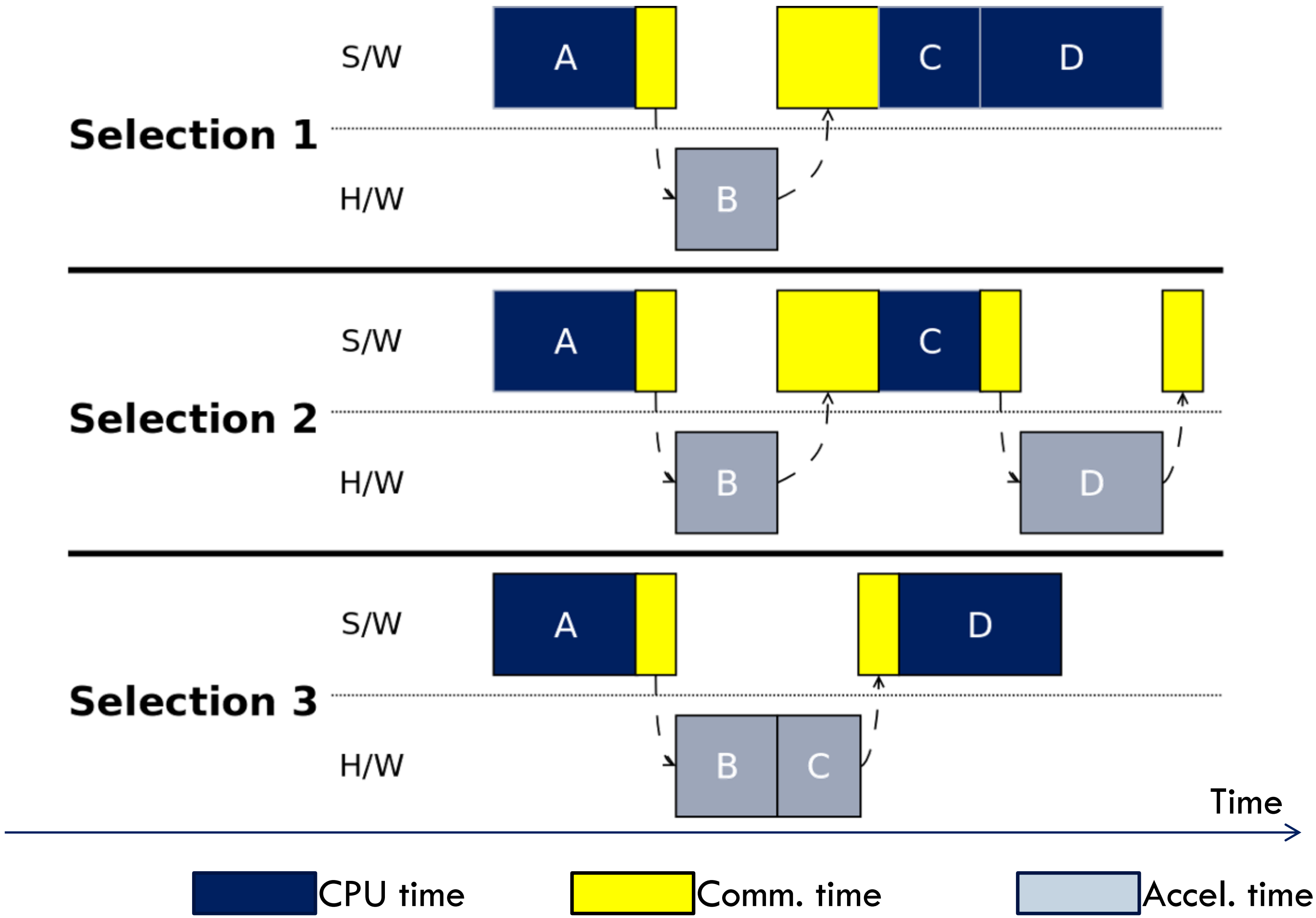


Platform-independent metrics

- Accelerator time and communication time are implementation-dependent!
  - Large design space for implementations

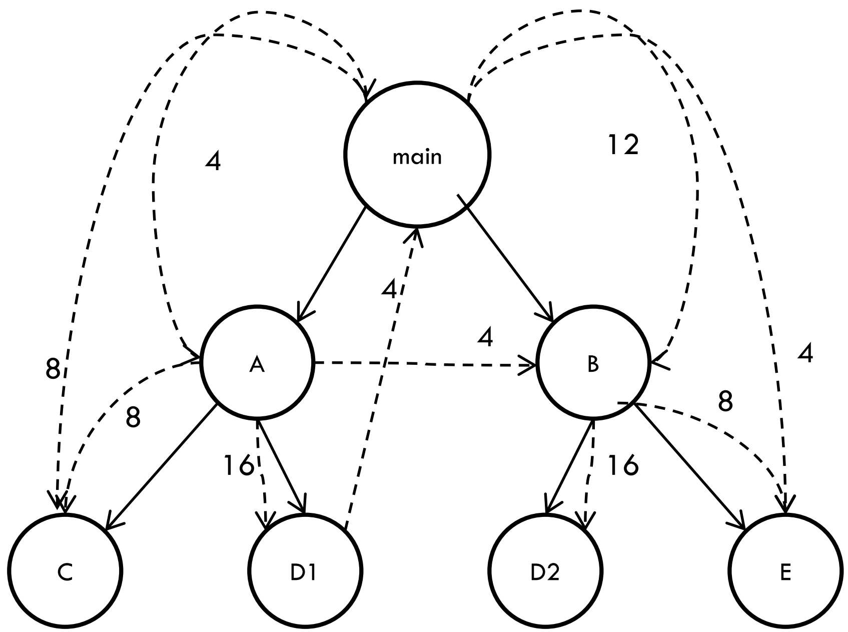
- Early stage design approach: Capture platform-independent metrics as proxy
  - Accelerator time → Compute operations
  - Communication Time → I/O set of bytes for each function

Accelerator selection



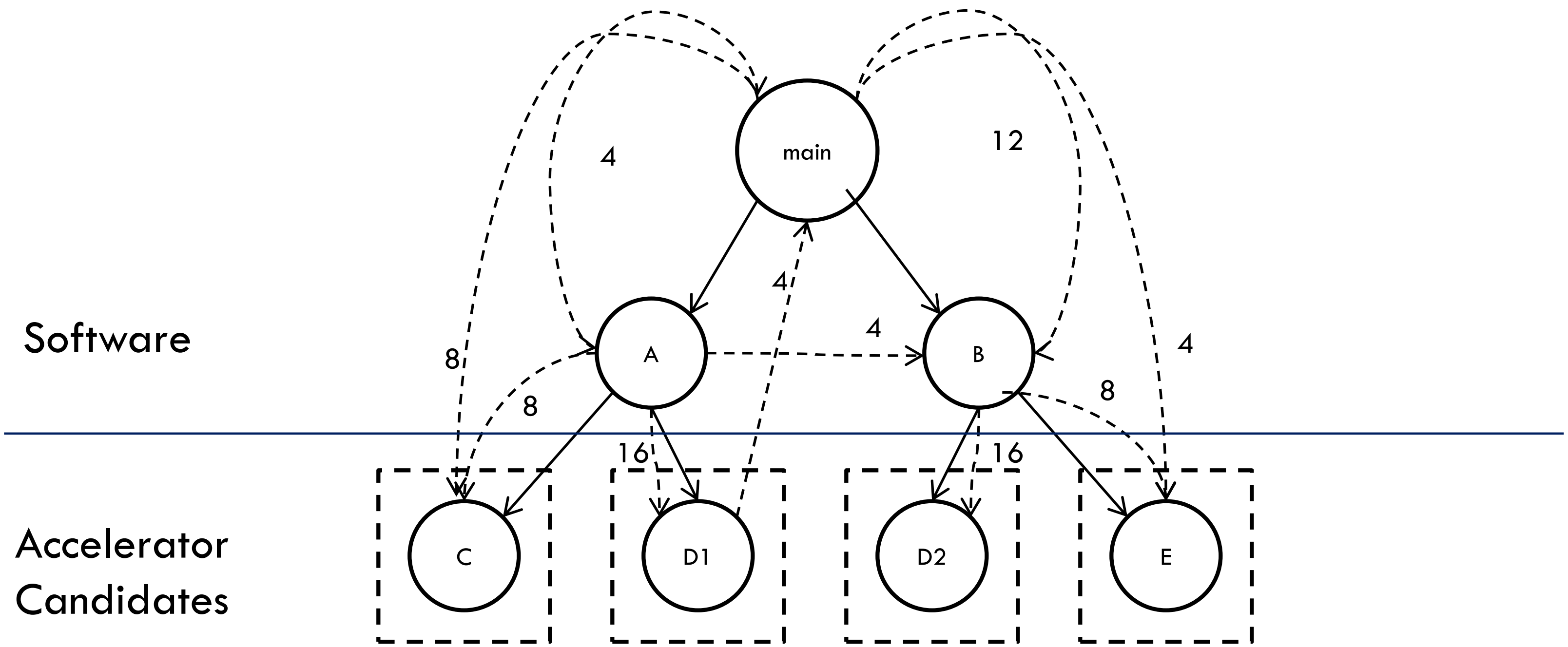
HW/SW partitioning process

- How to pick accelerator candidates in hierarchical CDFG?



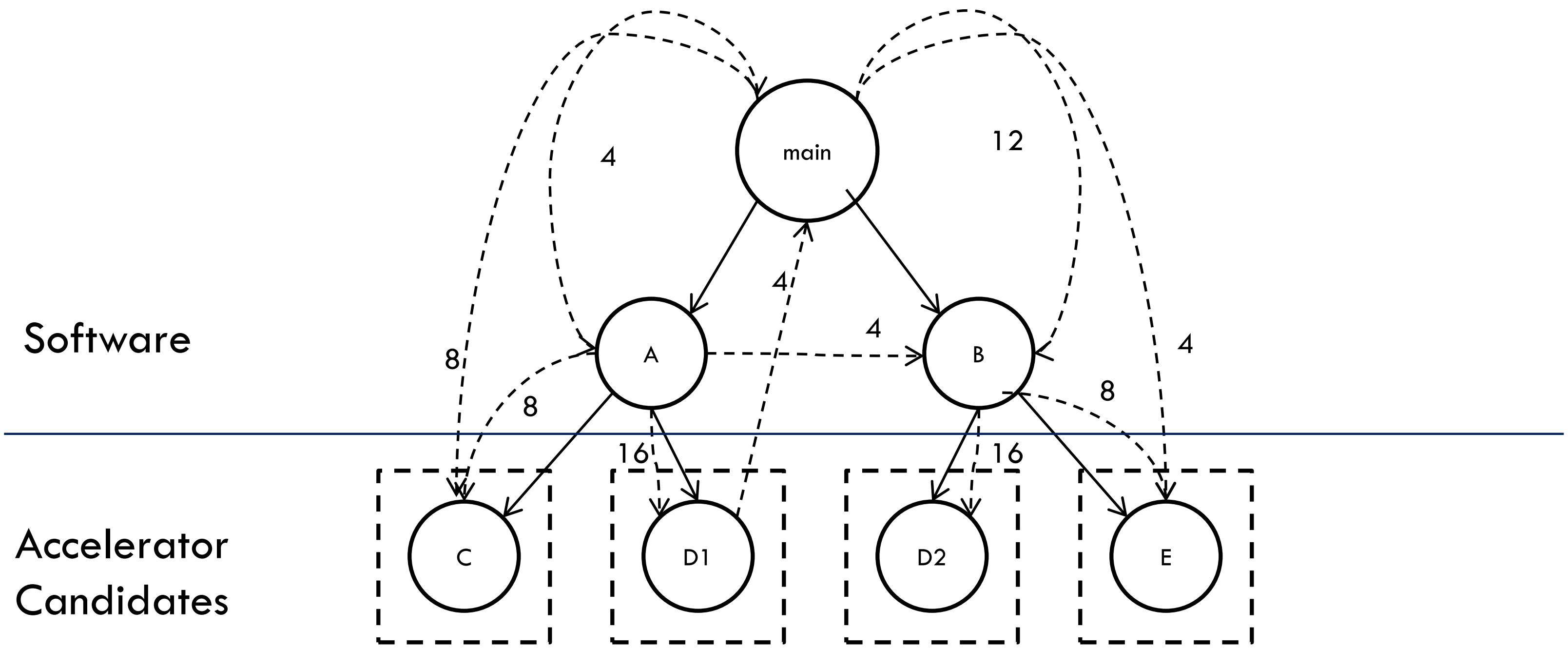
HW/SW partitioning process

- How to pick accelerator candidates?
  - Leaf nodes are self contained – Natural candidates
    - If coverage of work too low?



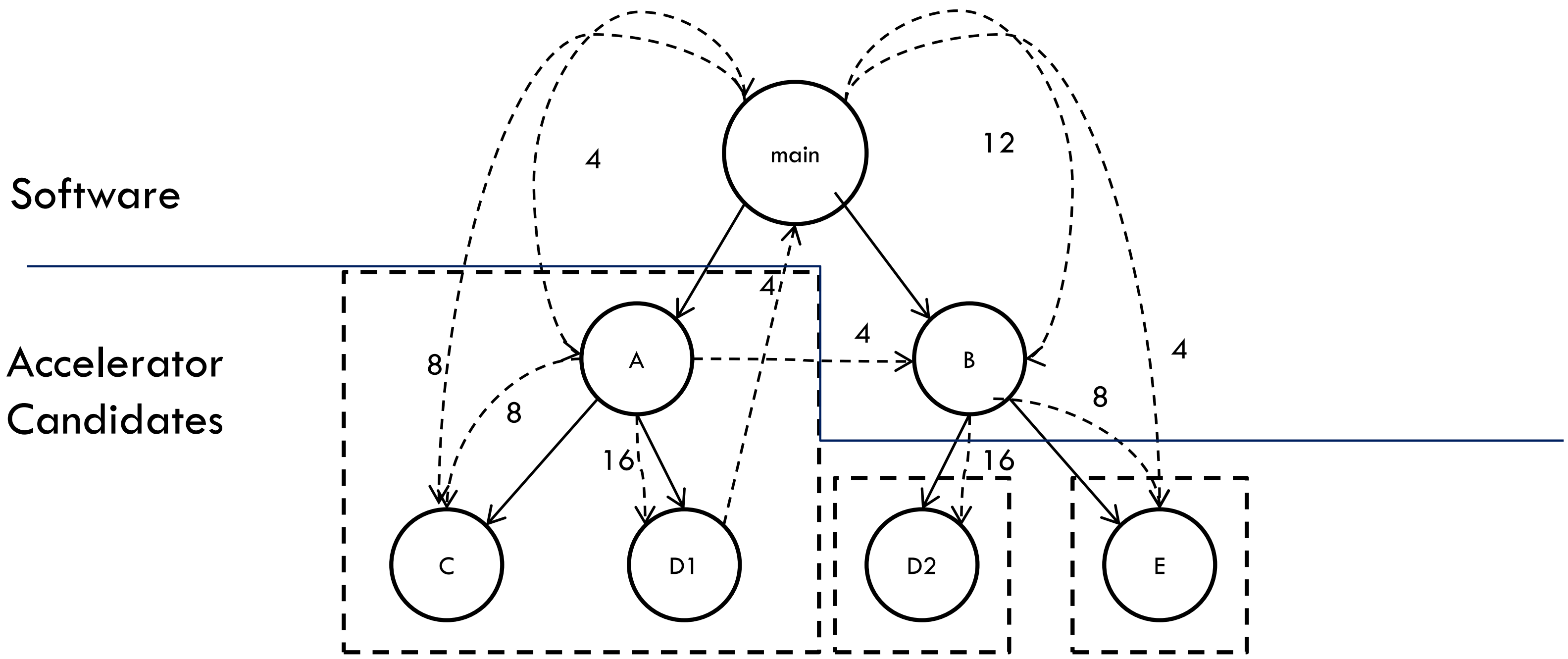
HW/SW partitioning process

- How to pick accelerator candidates?
  - Leaf nodes are self contained – Natural candidates
  - Non-leaf nodes? ***Include*** functionality of sub-calltree



Calculate inclusive costs

- Non-leaf nodes: Merge sub-calltree
  - *Inclusive computation costs – Add up operations*
  - *Inclusive communication costs – Edges crossing the box*



Outline

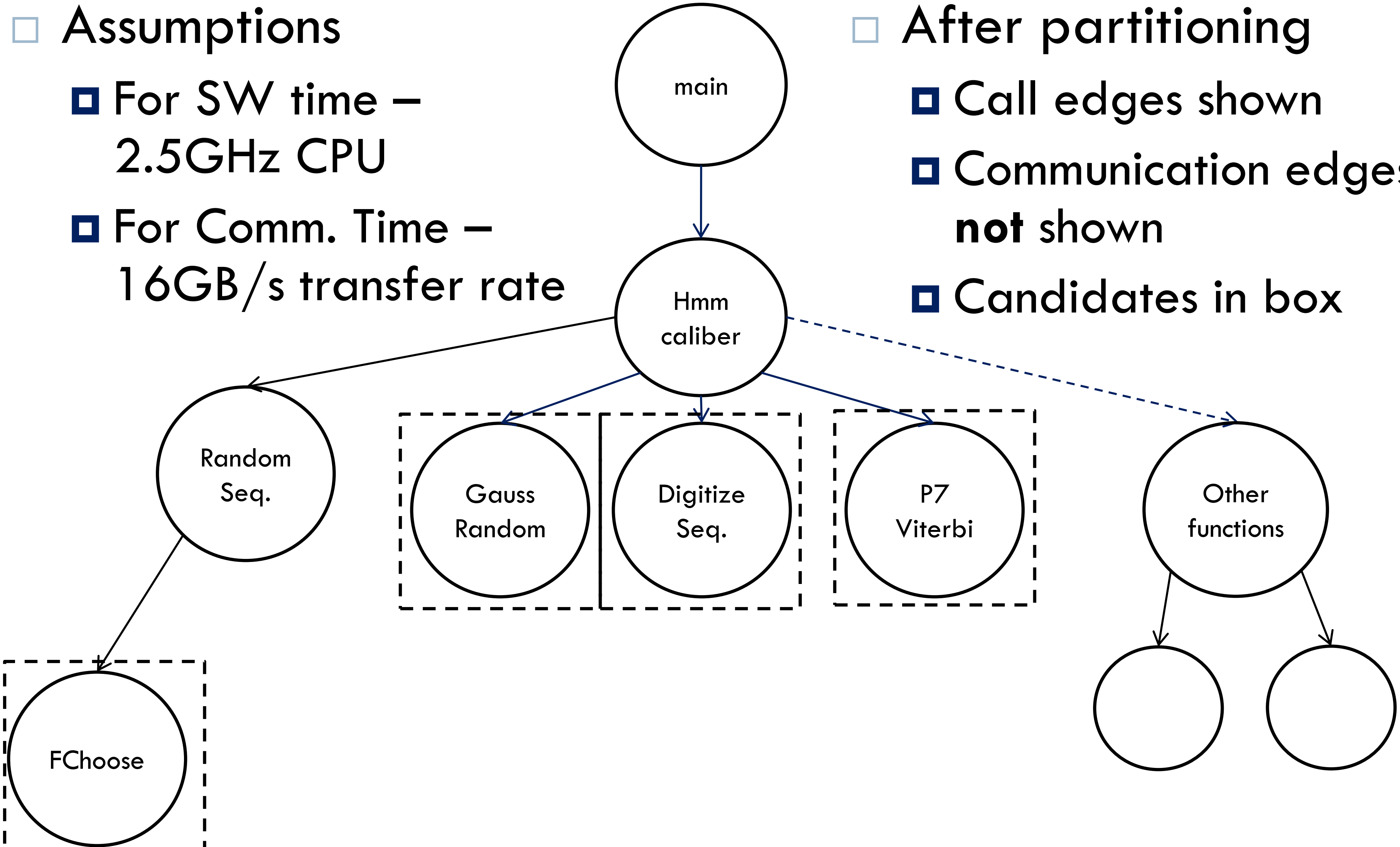
- Accelerator selection problem
- Sigil Overview
- Sigil methodology for accelerator selection
- **Partitioning examples**
  - In-depth look: 456.Hmmer
  - Results: Multiple benchmarks
- Building and Running Sigil

Partitioning algorithm

- Employ any partitioning algorithm
  - Existing algorithms
    - Intuitive: Computation to Communication ratio
    - State-of-the-art: Simulated Annealing, Genetic algorithms
  - We use a ***demonstrative*** algorithm utilizing:
    - software time – from Callgrind
    - communication time – from Sigil
    - compute time – from Sigil
  - Does not indicate amenability of functions
    - HLS tools show amenability

Partitioning example: Spec 456.Hmmer

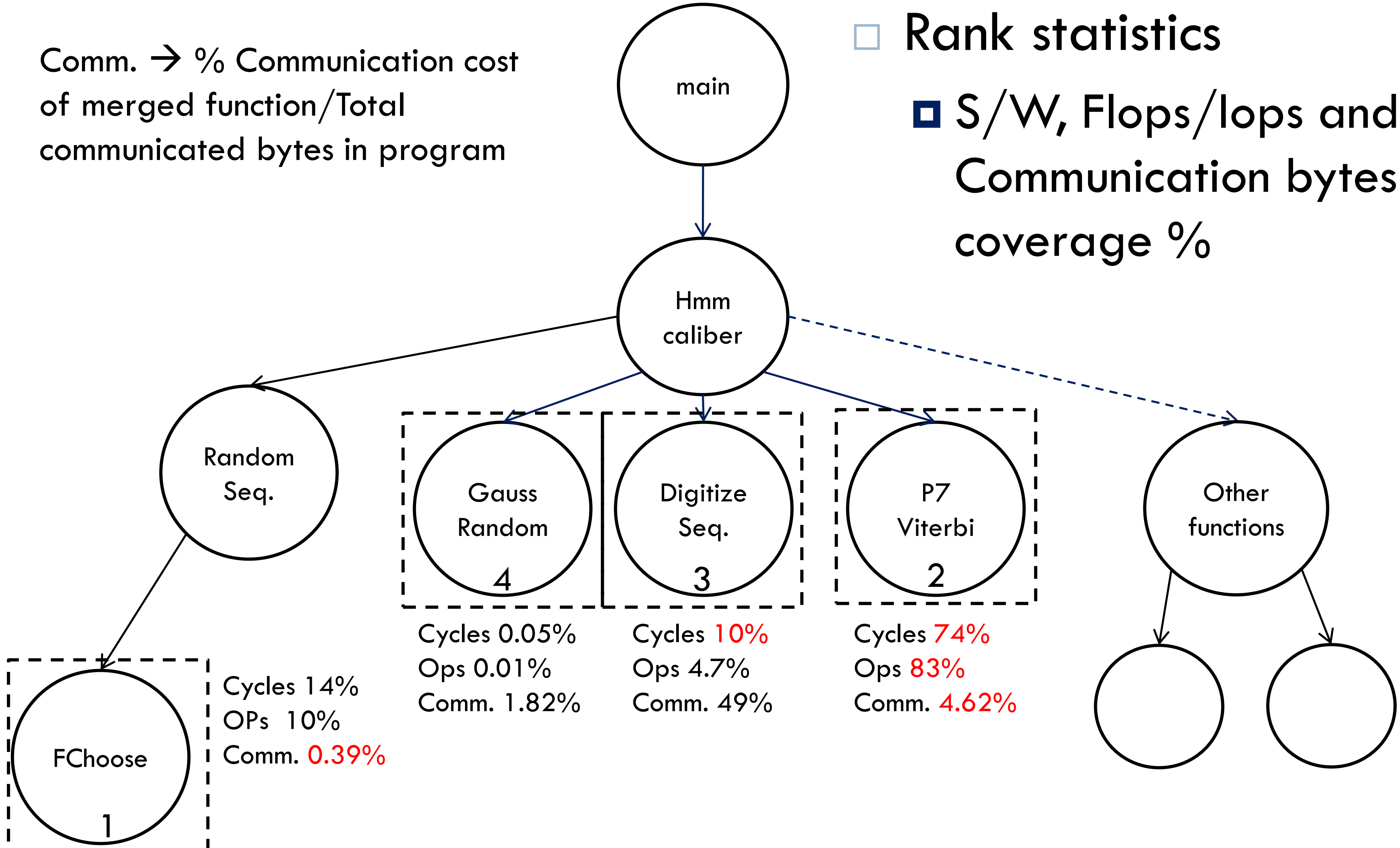
- Assumptions
  - For SW time – 2.5GHz CPU
  - For Comm. Time – 16GB/s transfer rate
- After partitioning
  - Call edges shown
  - Communication edges **not** shown
  - Candidates in box



Partitioning example: Spec 456.Hmmer

Comm. → % Communication cost of merged function/Total communicated bytes in program

- Rank statistics
  - S/W, Flops/Iops and Communication bytes coverage %



Partitioning Results - PARSEC

Functions from **demonstrative** partitioning for PARSEC benchmarks

Rank	Blackscholes	Freqmine	Dedup
1	String to float	sort	sha1_block_data_order
2	ieee754_exp	FP_Array_scan2*	sha1_block_data_order
3	ieee754_expf	sort	compress2*
4	ieee754_logf	FP_Array_scan2*	write_file*

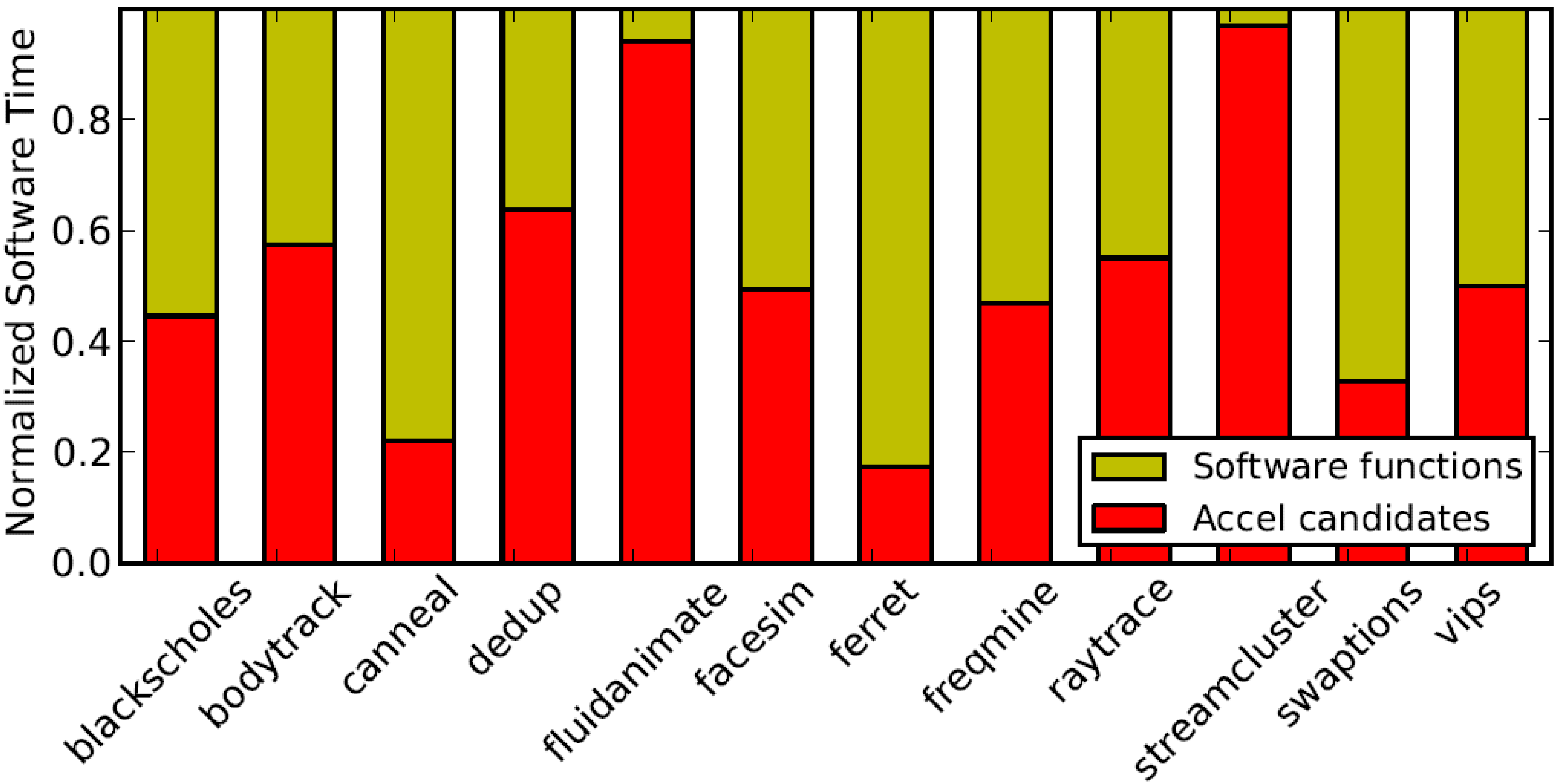
- ieee_754/mul – IEEE "math" library functions
- sha1_block_data_order – core of SHA1 calculation
- FP_Array_scan2 – Builds "prefix-tree" for frequent pattern mining [1]

* → merged function

Source: [1] C. Bienia et al. "The PARSEC Benchmark Suite: Characterization and Architectural Implications", Princeton Technical Report

Partitioning Results - PARSEC

- S/W Coverage with accelerator candidates



Outline

- Accelerator selection problem
- Sigil Overview
- Sigil methodology for accelerator selection
- Partitioning examples
- **Building and Running Sigil**

Getting Sigil

- Available open source
  - **git clone** https://github.com/snilakan/Sigil
  - Documentation included

- Tested and validated in Linux
  - Officially tested distros: CentOS6, Ubuntu 12.04 LTS, Ubuntu 14.04 LTS
  - Supported by any system supported by Valgrind (3.10.1)

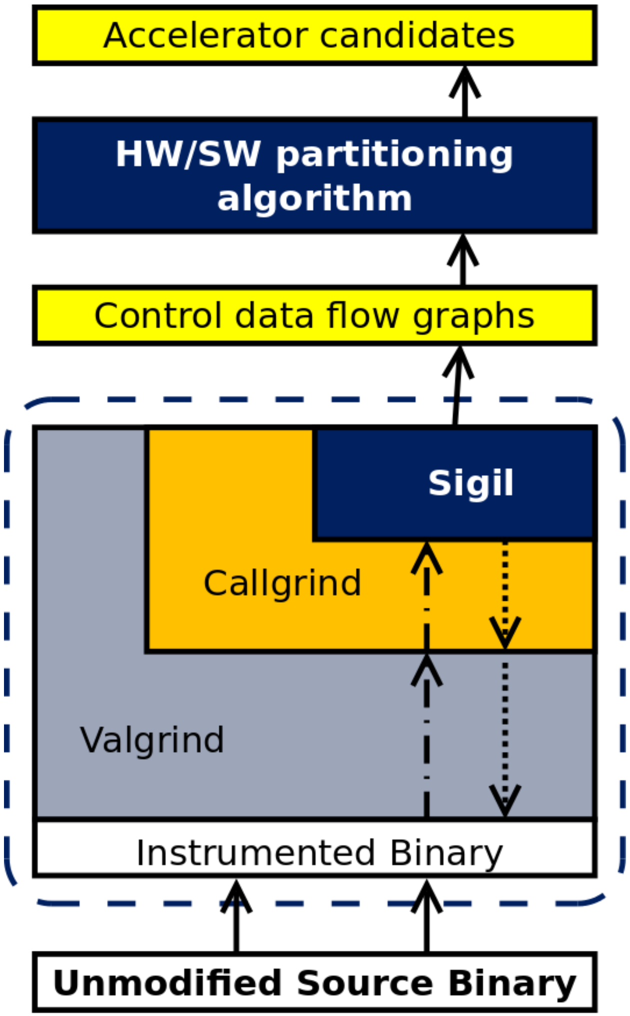
Building Sigil

- Automated script
  - Checks dependencies and builds Valgrind with Sigil/Callgrind
  - Configures post processing scripts

- Manual build process
  - Autotools build process – basically building Valgrind
    - `$ ./autogen.sh`
    - `$ ./configure`
    - `$ make`
  - Small path modifications in post processing scripts
    - See documentation

Running Sigil

- Compile user program with **debug flags**
- Generate **CDFGs**
  - `$ ./run_sigil.sh my_binary`
  - Outputs sigil.totals.out-#
    - Thread #
- Partitioning the graph
  - Post-processing not part of Sigil
  - `$ ./aggregate_costs.py -help`
  - Our example partitioning algorithm
  - Can plug in your own partitioning algorithm!



Running Sigil - Caveats

- Before we begin…
- `$ ./run_sigil.sh`
  - Just a wrapper for typical usage of Sigil
  - May have to tweak built-in options
    - e.g. --separate-callers=#
      - Essentially specifies max nested function calls
      - Callgrind option
- Bounded by (Val/Call)grind's abilities
  - Usually memory allocation problems, if any at all
- ***Now, don't be a stranger!***
  - *Please contact us with issues or suggestions!*

Sigil Output

- Sample output from FFT kernel in Parsec 3.0 / SPLASH2x
- …top of file

```
SUMMARY:

Total Memory Reads(bytes): 329263019          Total Memory Writes(bytes): 192147336

Num SMs: 67                    Num funcinsts: 525                  Memory for SM(bytes): 281



TREE DUMP

FUNCTION NUMBER, FUNC_INST NUM, Children?, Number of calls
1, 0, True, 1
2, 0, True, 1
3, 0, False, 1
4, 0, True, 1
5, 0, False, 1
6, 0, False, 1
7, 0, False, 1
8, 0, True, 1
9, 0, False, 2
```

Sigil Output

- Sample output from FFT kernel in Parsec 3.0 / SPLASH2x
- …meanwhile way below…

```
DATA DUMP

THREAD NUMBER        FUNCTION NUMBER      FUNC_INST NUM        FUNCTION NAME


1                    1                    0                    0x0000000000001420
0                    30000                0                    NO PRODUCER
1                    19                   0                    _dl_add_to_namespace_list

1                    158                  0                    0x00000000004019ac
1                    102                  0                    _dl_init


1                    2                    0                    _dl_start
0                    30000                0                    NO PRODUCER
0                    20000                0                    SELF

1                    67                   2                    do_lookup_x
1                    4                    0                    _dl_sysdep_start
1                    65                   0                    _dl_relocate_object
1                    67                   0                    do_lookup_x
1                    60                   0                    _dl_check_map_versions
1                    61                   0                    match_symbol
```

Sigil Output

- Sample output from FFT kernel in Parsec 3.0 / SPLASH2x
  - Interesting, but not very clear on its own

- Gives us:
  - Communication edges
  - Classified communication counts
  - Compute counts
  - Some tool usage stats

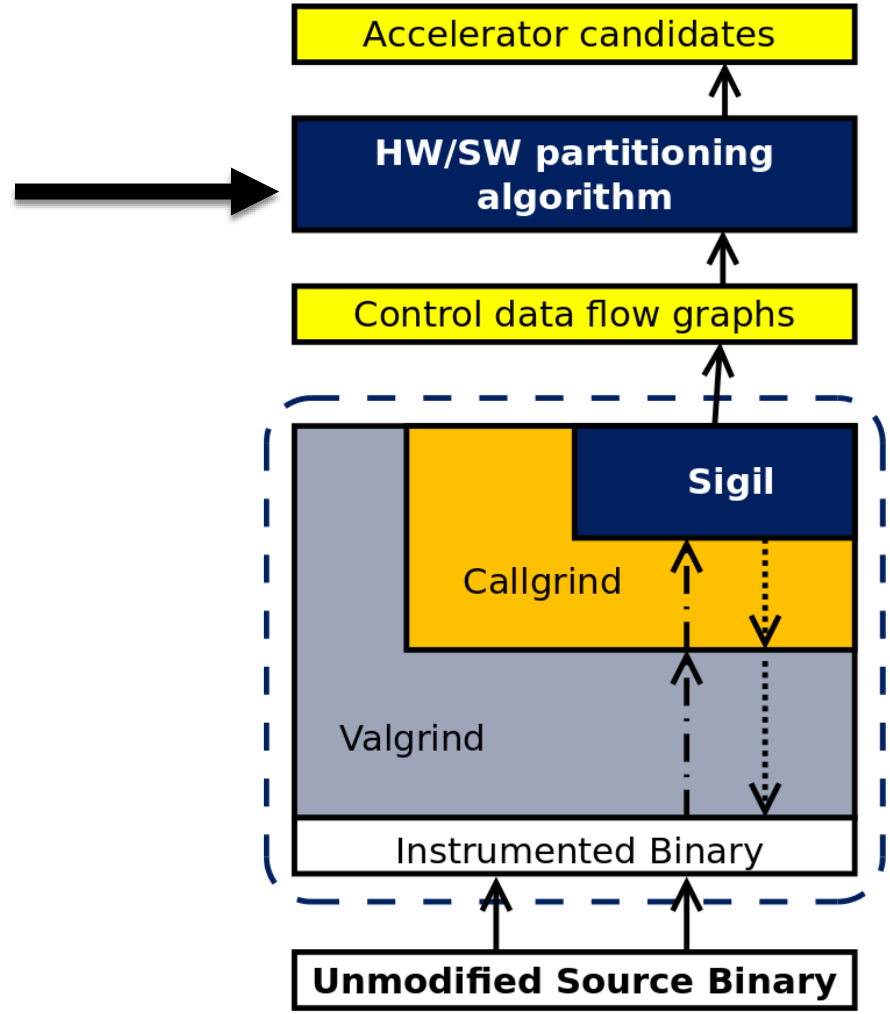
Post Processing

- Let's do something with this data!
- Partition call-tree (from Callgrind) with communication and computation costs (from Sigil)

  - Create call-tree (with Callgrind)
    - $ vg-in-place --tool=callgrind --cache-sim=yes --branch-sim=yes **my_binary**
  - Read in the data and make partitioning choices

Post Processing

Remember me?



Post Processing

- Let's do something with this data!
- Partition call-tree (from Callgrind) with communication and computation costs (from Sigil)

  - Create call-tree (with Callgrind)
    - $ vg-in-place --tool=callgrind --cache-sim=yes --branch-sim=yes **my_binary**
  - Read in the data and make partitioning choices
    - Our demonstrative partitioning script is included
    - Example use:
      - $ ./aggregate_costs_gran.py .../sigil.totals.out-1 --trim-tree --cg-file=.../callgrind_output_file --gran-mode=metric > my_postprocessed_workload.txt

Post Processing

- Let's do something with this data!

```
240         0          InitX                                               * 1          34873399       0
247         0          drand48                                             * 524288     31981568       0
248         0          erand48_r                                           * 524288     27787264       0
249         0          __drand48_iterate                                   * 524288     10485760       0
250         0          InitU                                               * 1          137110         515
253         0          sincos                                              * 512        126645         0
257         0          __cos_avx                                           * 512        57930          0
255         0          __sin_avx                                           * 512        55915          0
192         0          malloc                                              * 7          58780          0
193         0          malloc_hook_ini                                     * 1          57592          0
194         0          ptmalloc_init.part.8                                * 1          56386          0
195         0          _dl_addr                                            * 1          56056          0
226         0          printf                                              * 13         19130          0
227         0          vfprintf                                            * 13         18841          0

Trimming tree.....
Func num    Func Inst  Function name                                        Numcalls   Instrs          Flops

Uppertree Software Time (Cycles): 103725108.000000
Bottom nodes Software Time (Cycles): 232096483.000000

247         0          drand48                                             * 524288     31981568       0
253         1          sincos                                              * 262144     63651539       0
275         0          FFT1DOnce.constprop.2                               * 1024       109542372      0
250         0          InitU                                               * 1          137110         515
272         0          Transpose                                           * 3          4277835        0
195         0          _dl_addr                                            * 1          56056          0
~
```

Post Processing

- Let's do something with this data!

```
Trimming tree.....
Func num    Func Inst   Function name                                     Numcalls    Instrs          Flop

Uppertree Software Time (Cycles): 103725108.000000
Bottom nodes Software Time (Cycles): 232096483.000000

247         0           drand48                                          * 524288      31981568        0
253         1           sincos                                           * 262144      63651539        0
275         0           FFT1DOnce.constprop.2                            * 1024        109542372       0
250         0           InitU                                            * 1         137110          515
272         0           Transpose                                        * 3         4277835         0
195         0           _dl_addr                                         * 1         56056           0
~
```

- ...and we finally have our merged, leaf node candidates, from our demonstrative algorithm!

Looking forward

- We plan on releasing results from SPEC, PARSEC, BioBench and more
- Improving interface and documentation
  - Under the hood overhaul
- Commonality of functions between applications
  - Area may be free, design and verification are not
- Need more applications!
  - Run Sigil on your workload and tell us what you find

Wrap Up

- Available
  - **git clone** https://github.com/snilakan/Sigil
  - http://dpac.ece.drexel.edu/current-research-projects/sigil/

- Contact: michael.d.lui@drexel.edu

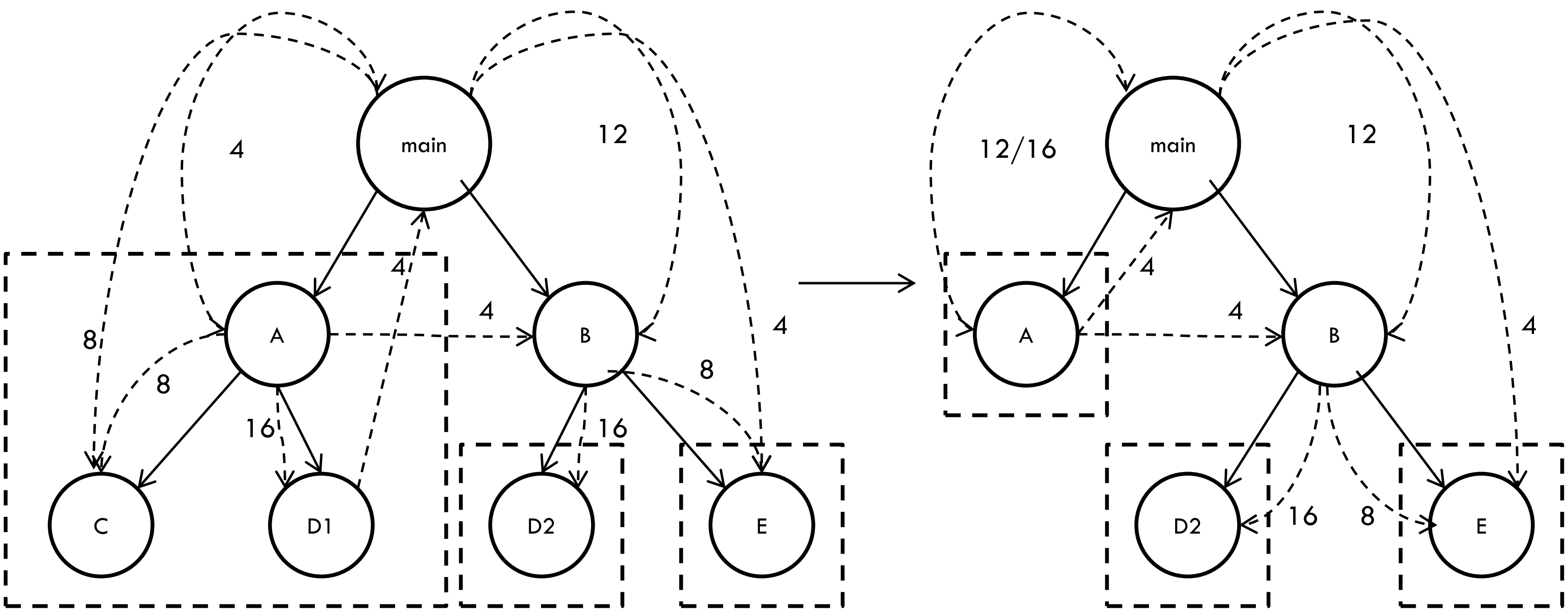
- Demo Later

- Related Publications
  - "Platform-independent Analysis of Function-level Communication in Workloads", Siddharth Nilakantan and Mark Hempstead, IISWC 2013
  - "Metrics for Early-Stage Modeling of Many-Accelerator Architectures", Siddharth Nilakantan, Steven Battle and Mark Hempstead, CAL July-Dec 2012

BACKUP SLIDES

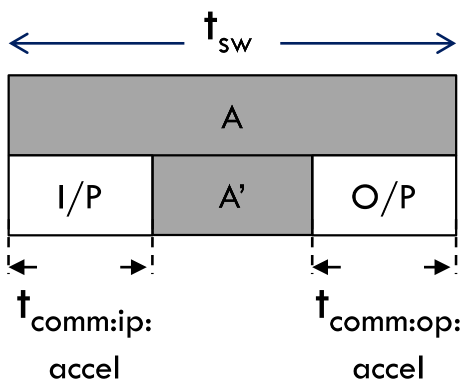
Partitioning steps

- First, use a metric to compare nodes against parents
  - Merge nodes when parents make better candidates
- Second, rank leaf nodes by same metric



Metric for merging & ranking

- Breakeven-speedup
  - Minimum factor of computational acceleration, given communication
  - For calculation of communication; we can plug in a transfer rate



Breakeven-speedup =

$$\frac{t_{sw}}{t_{sw} - (t_{comm:ip:accel} + t_{comm:op:accel})}$$