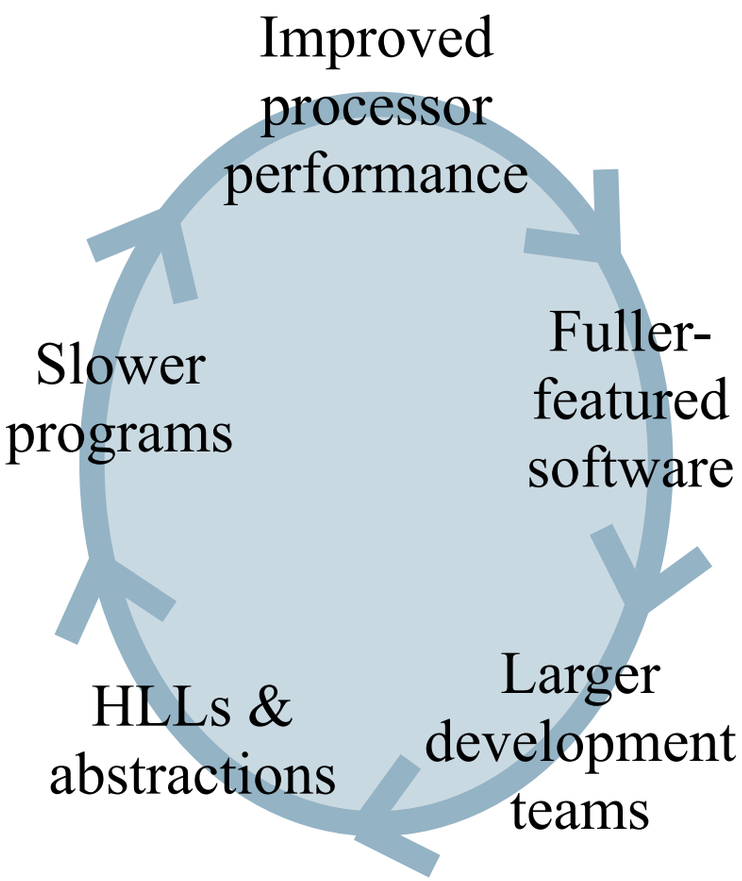
Computation vs. Memory Systems: Pinning Down Accelerator Bottlenecks

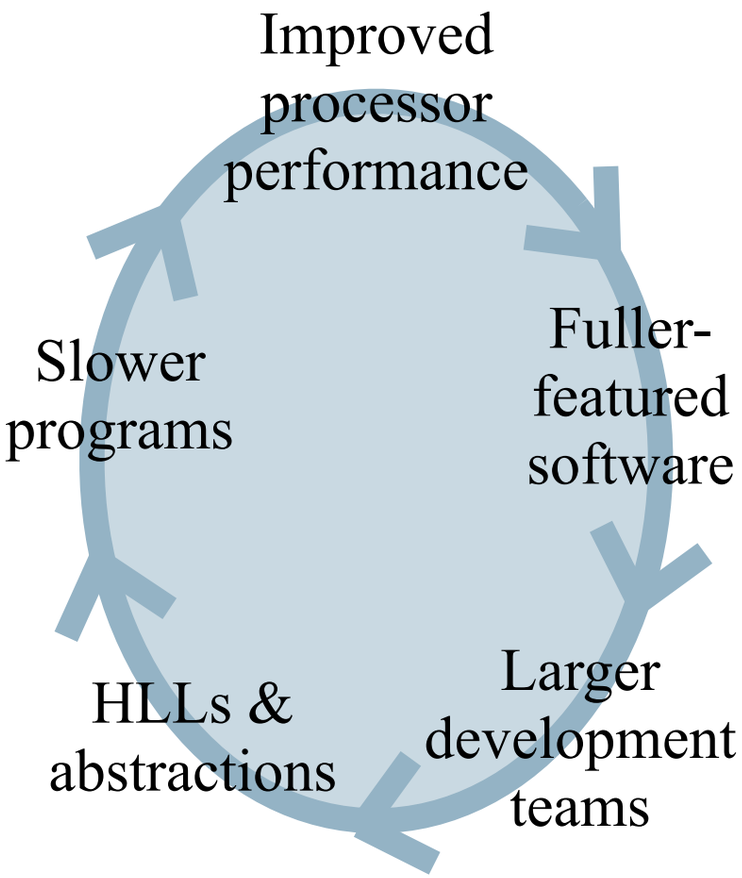
Martha Kim and Stephen Edwards

Columbia University
Department of Computer Science

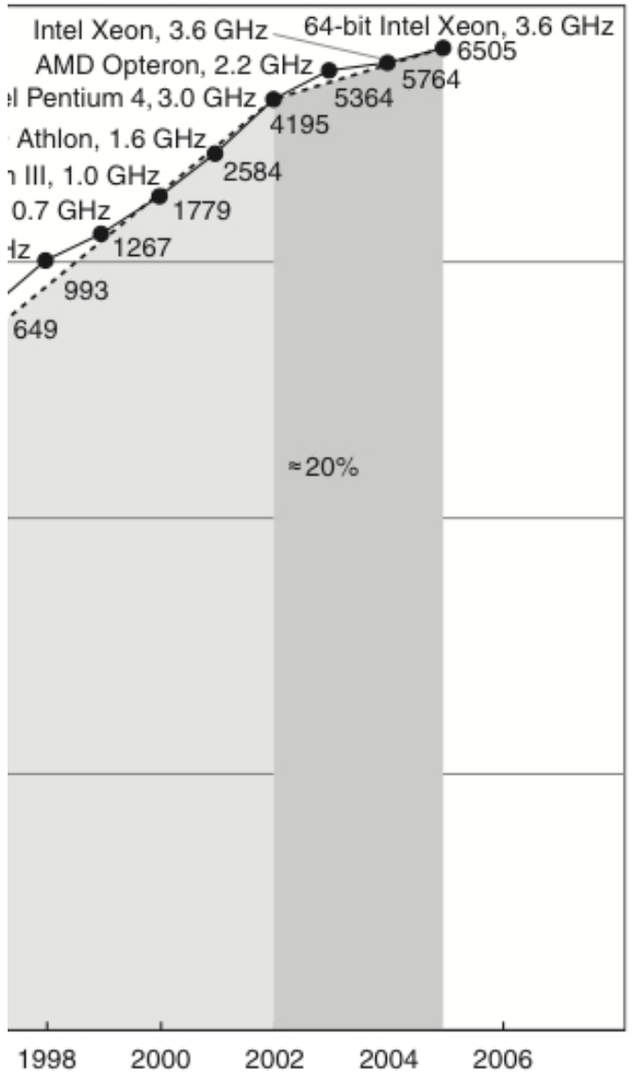
June 19, 2010
AMAS-BT Workshop



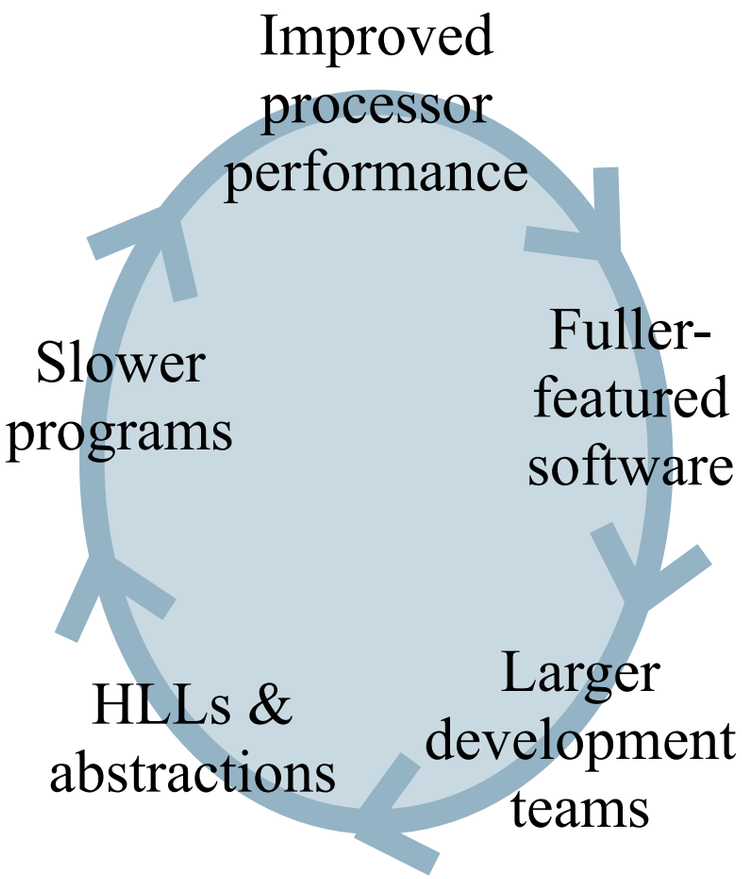
Larus' *virtuous cycle*



Larus' *virtuous cycle*

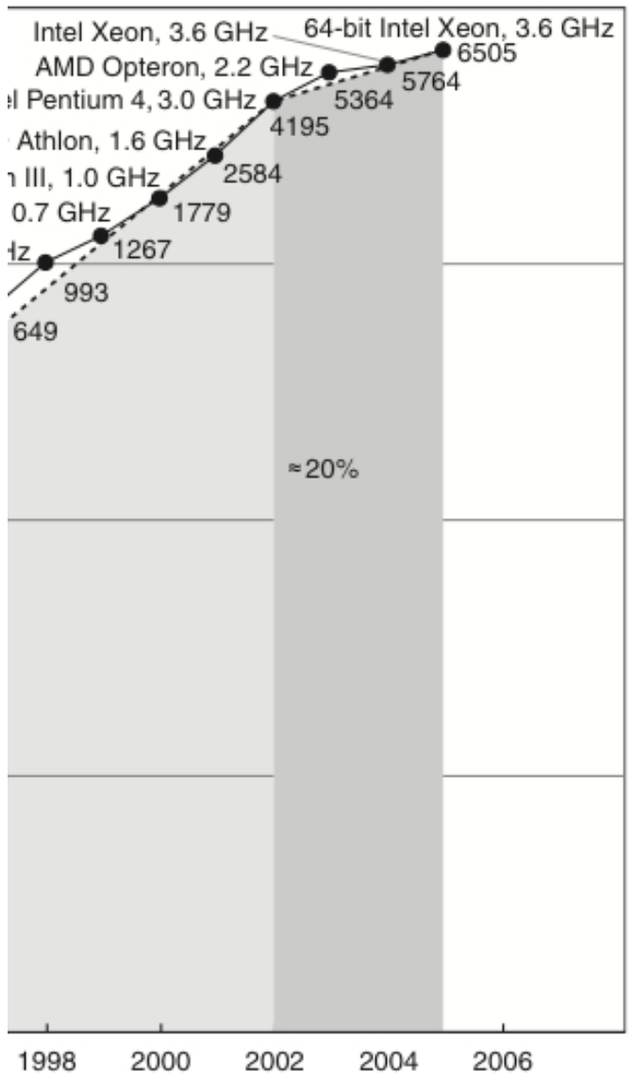


Power wall



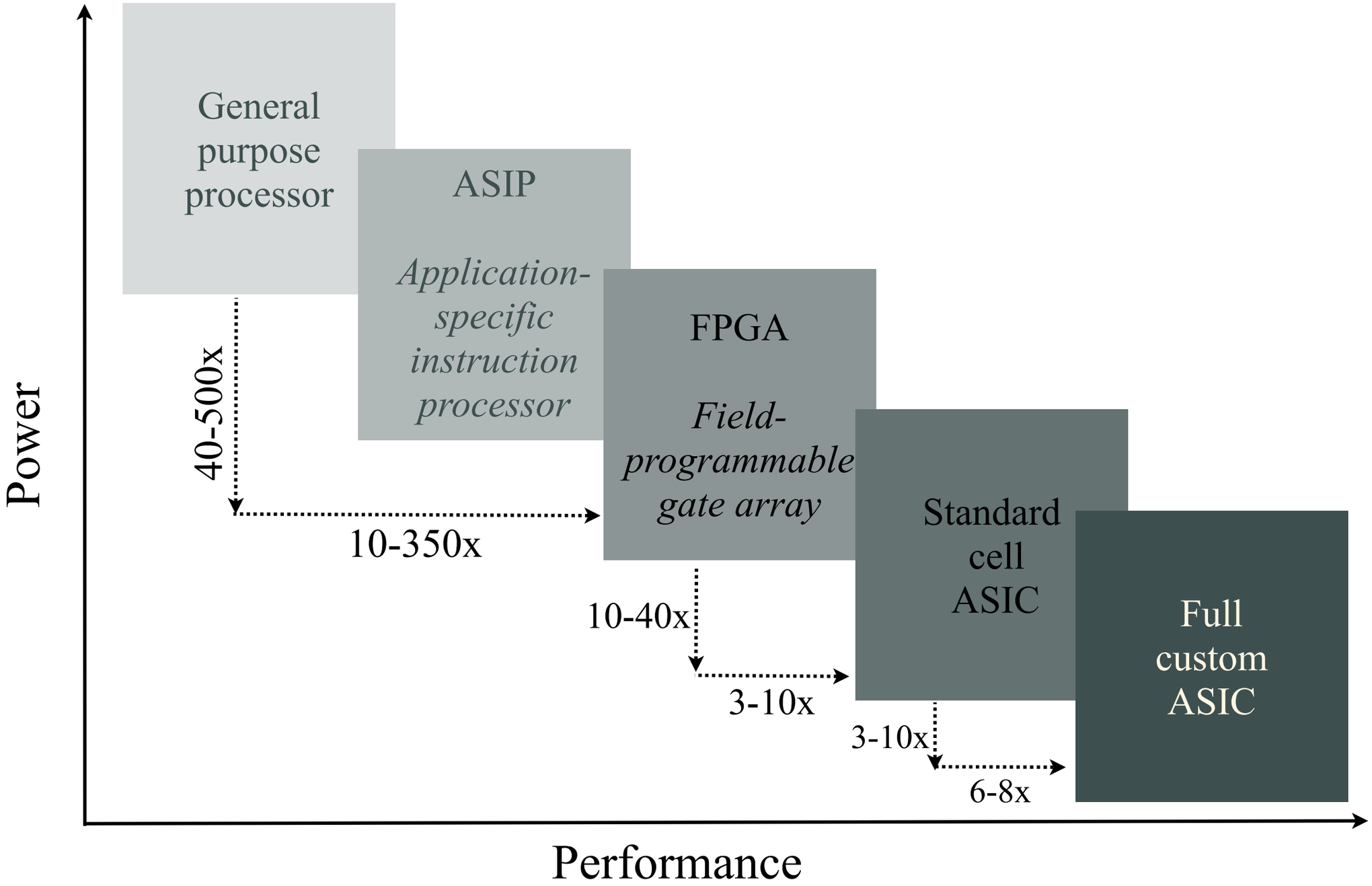
Larus' *virtuous cycle*

Increases in power efficiency **and** performance



Power wall

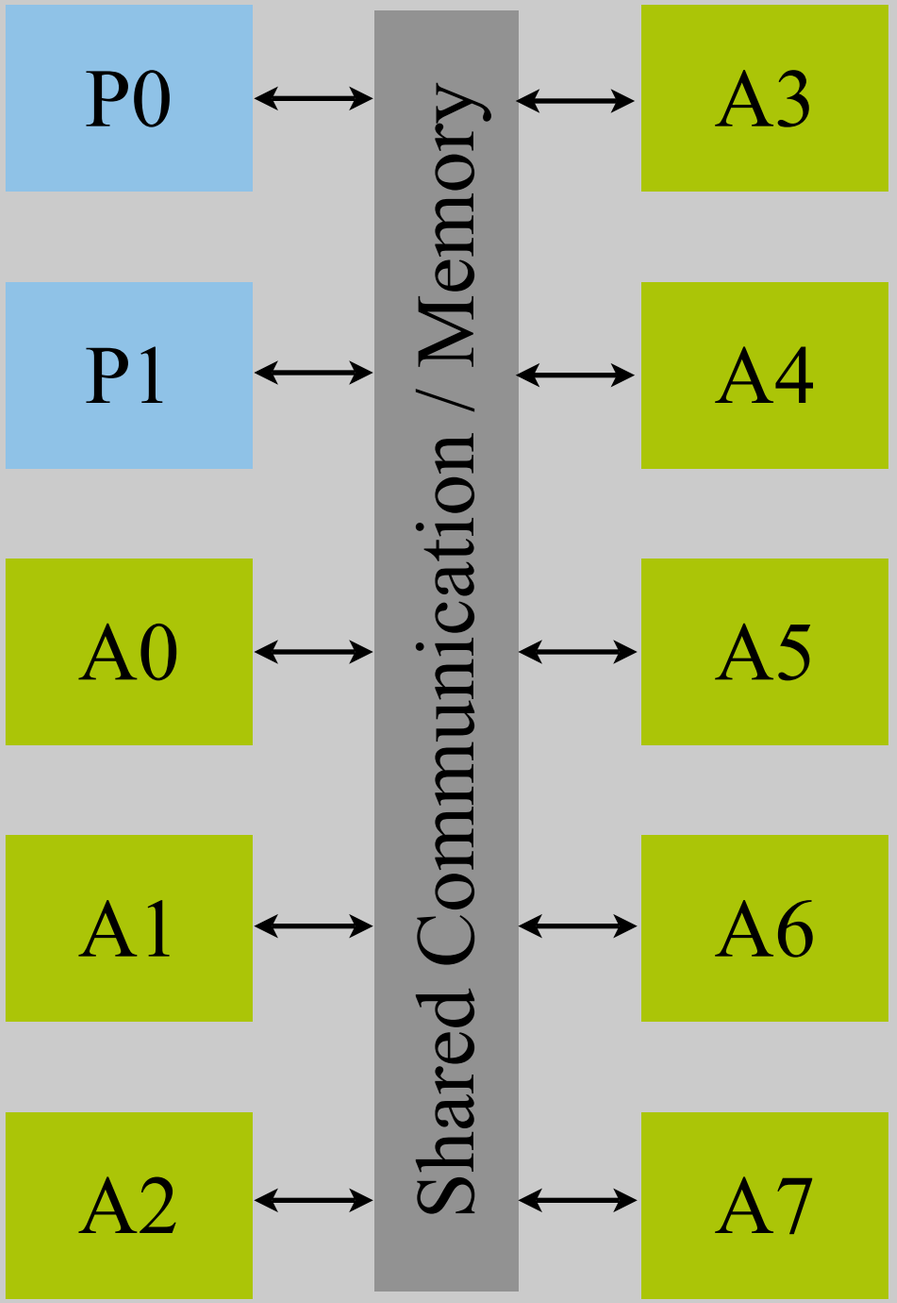
Efficiency of Specialized Hardware



Accelerator System

General-purpose core(s) surrounded by many, many special-purpose accelerators that are powered on only when their function is needed.

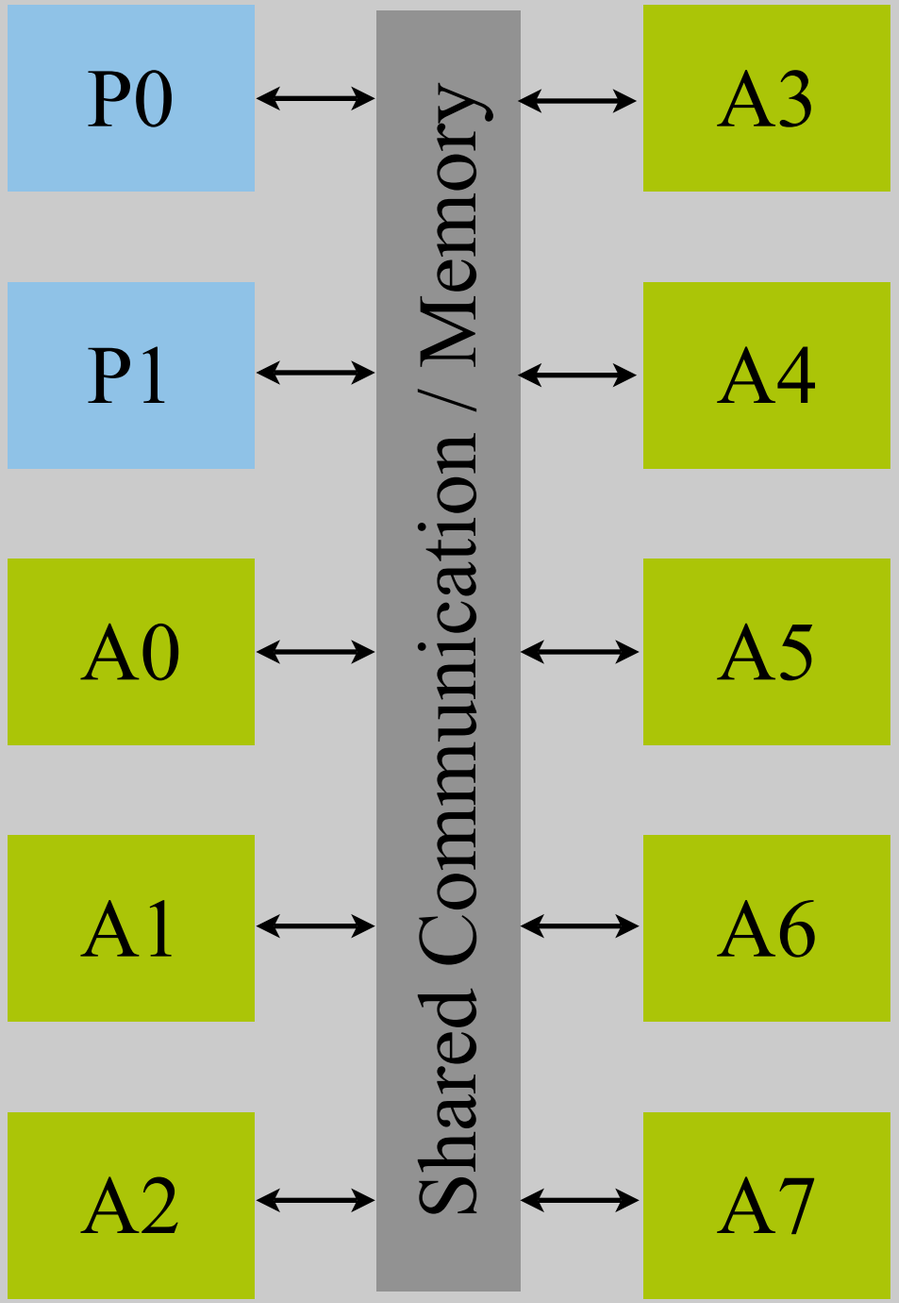
Potential benefits available only if applications actually make use of accelerators.



Accelerator System

General-purpose core(s) surrounded by many, many special-purpose accelerators that are powered on only when their function is needed.

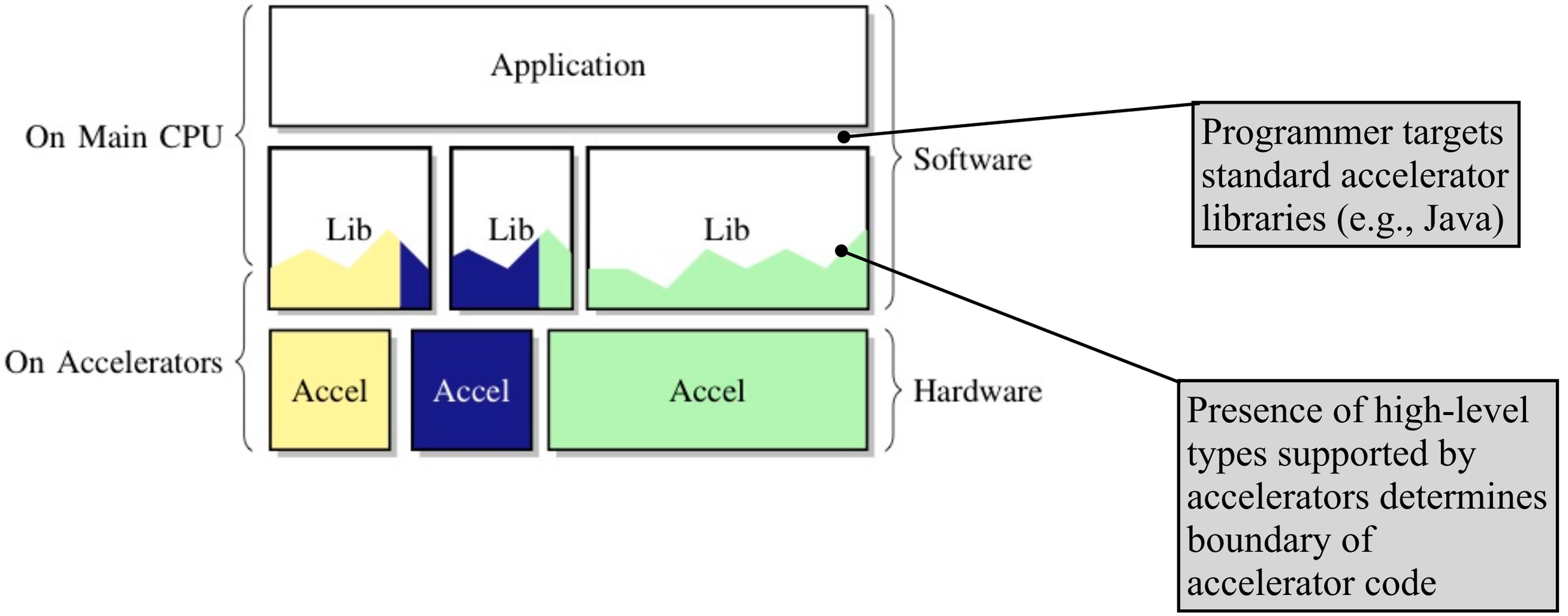
Potential benefits available only if applications actually make use of accelerators.



Talk Outline

- Overall Vision
- Accelerator System Model
- Methodology Overview
- Methodology in Practice
  - Image Rotation
  - JPEG
- Conclusion

System-Level Vision

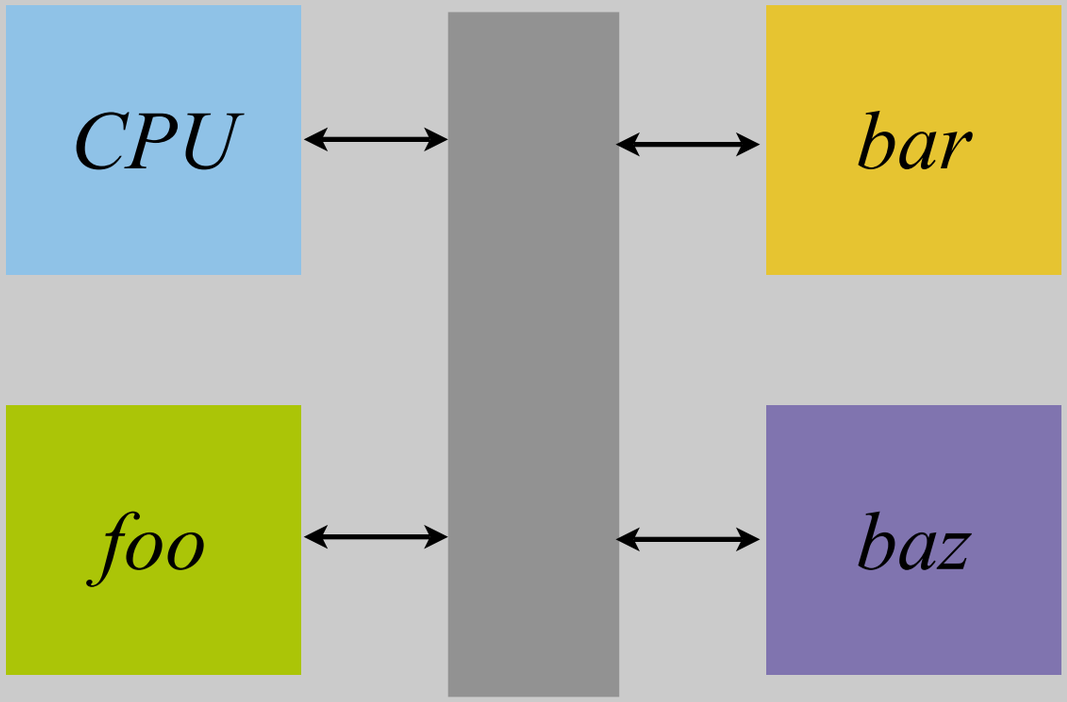


Functions define boundaries of acceleration.

Each accelerator must do two things

1. Compute

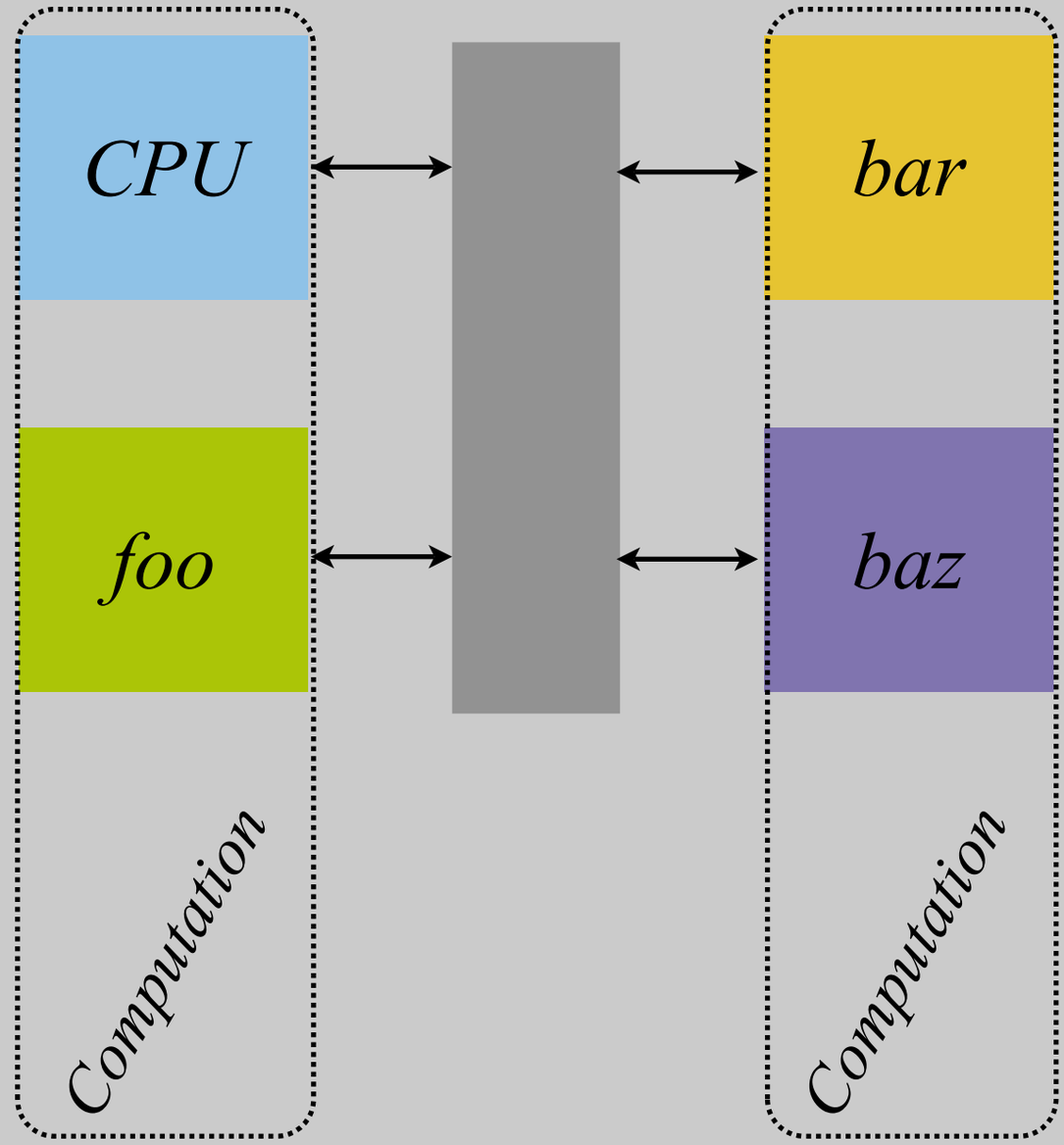
2. Communicate externally



Each accelerator must do two things

1. Compute

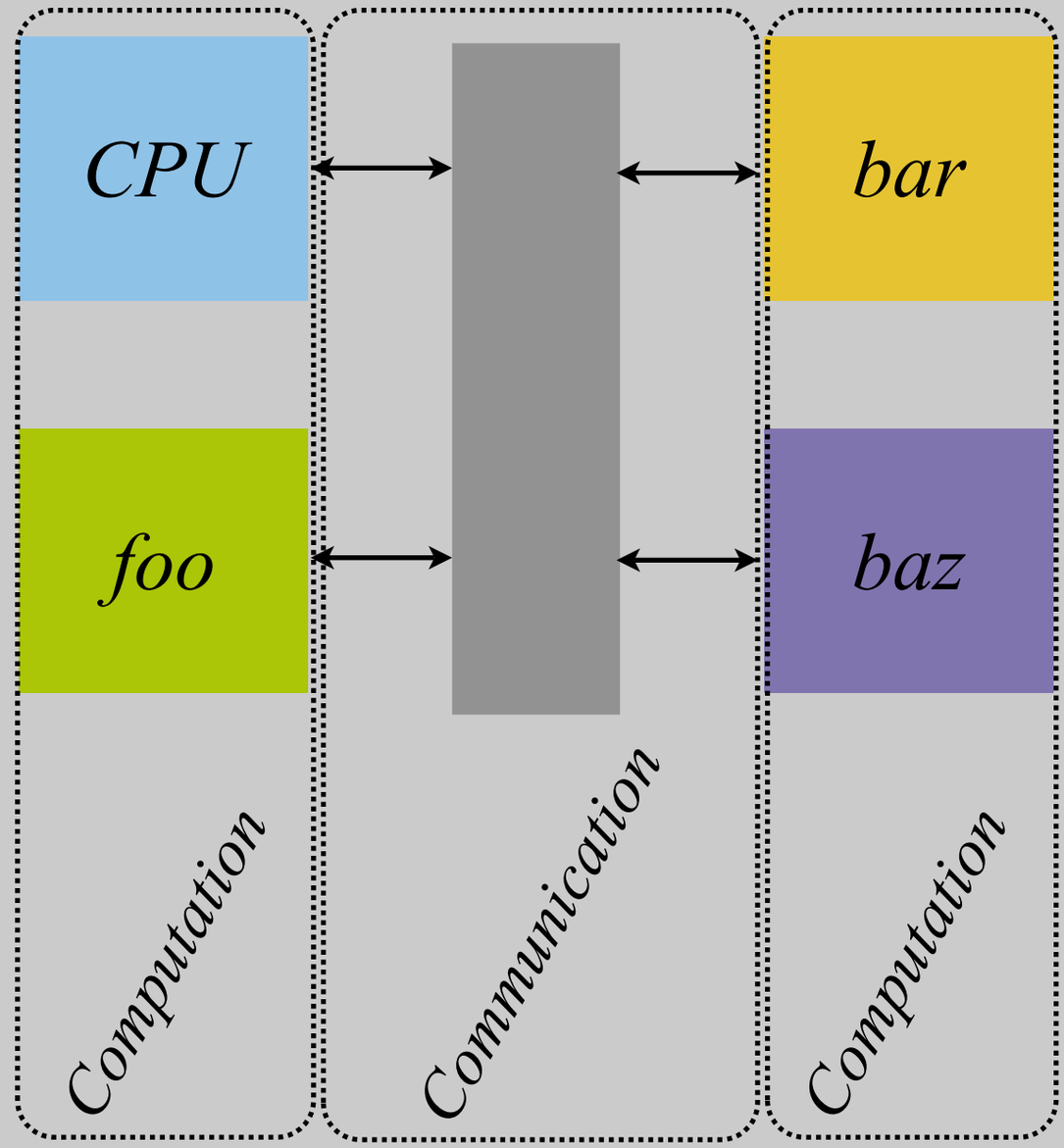
2. Communicate externally



Each accelerator must do two things

1. Compute

2. Communicate externally



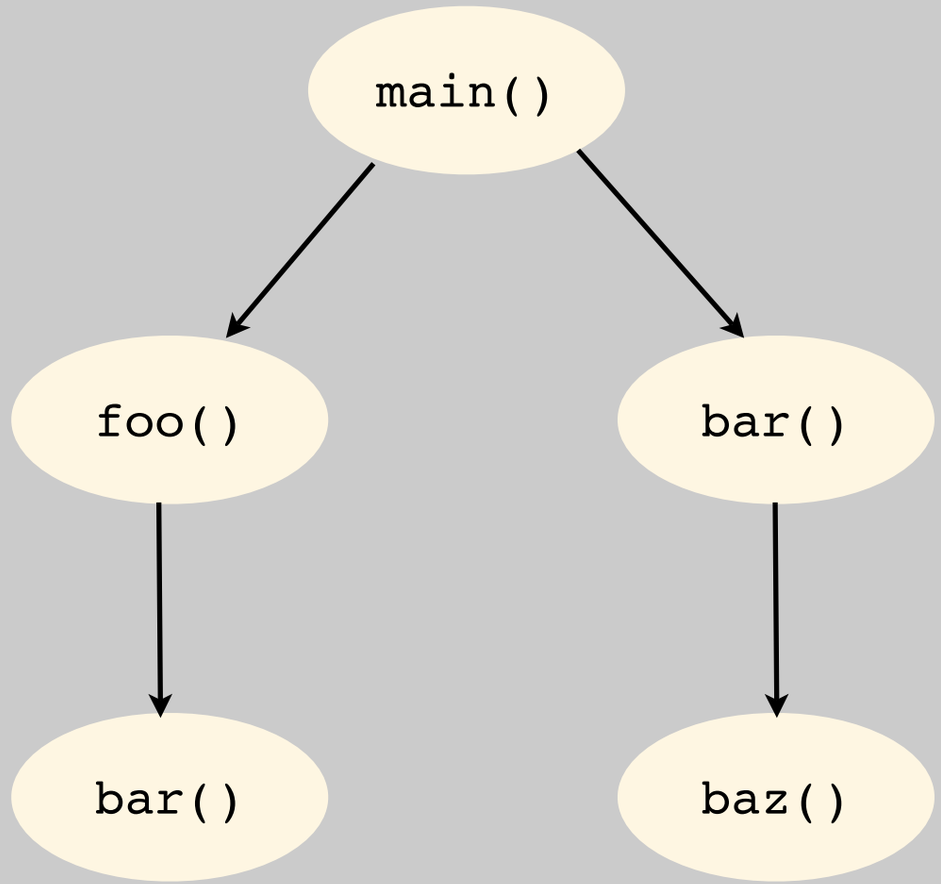
Talk Outline

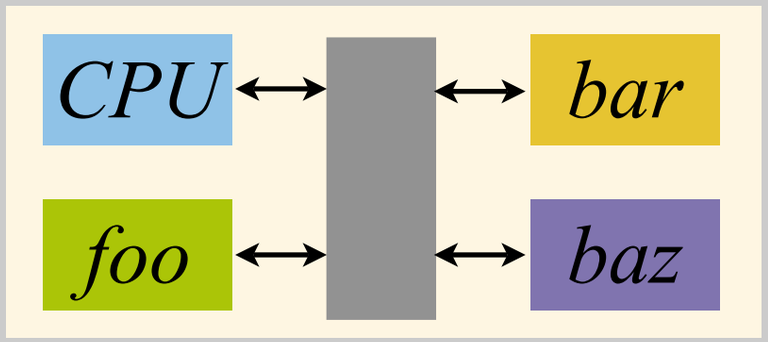
- Overall Vision
- Accelerator System Model
- Methodology Overview
- Methodology in Practice
  - Image Rotation
  - JPEG
- Conclusion

Computation Model

1. Dynamic function invocations mapped to corresponding accelerator

Dynamic invocation tree:

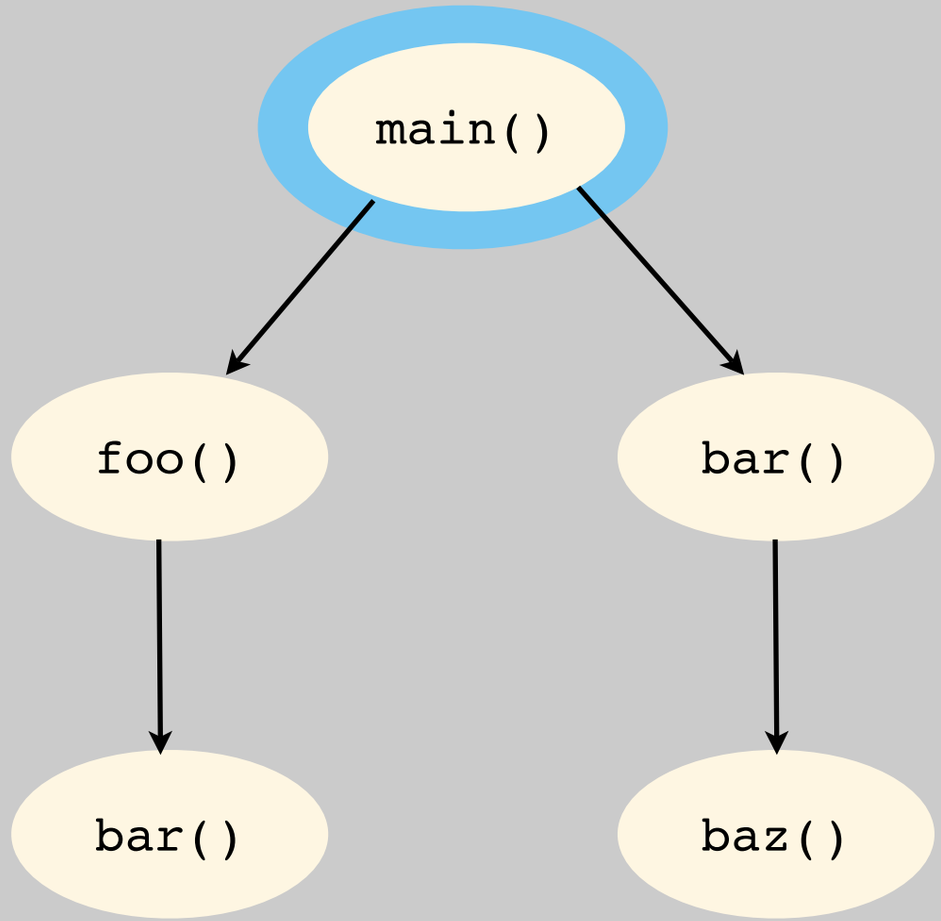


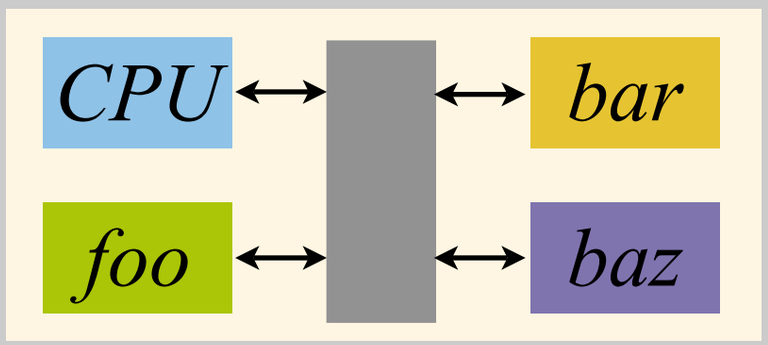


Computation Model

1. Dynamic function invocations mapped to corresponding accelerator

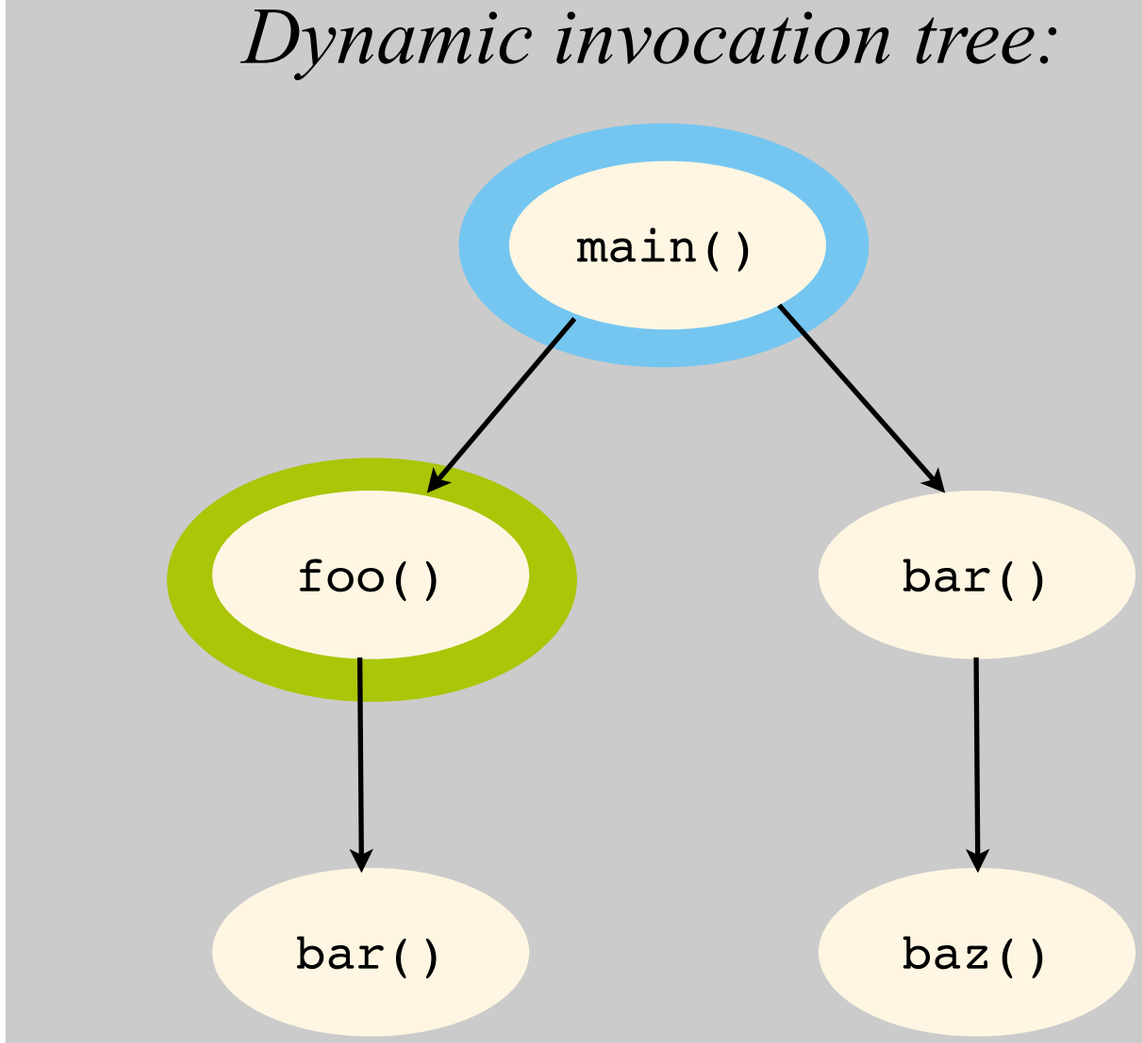
Dynamic invocation tree:

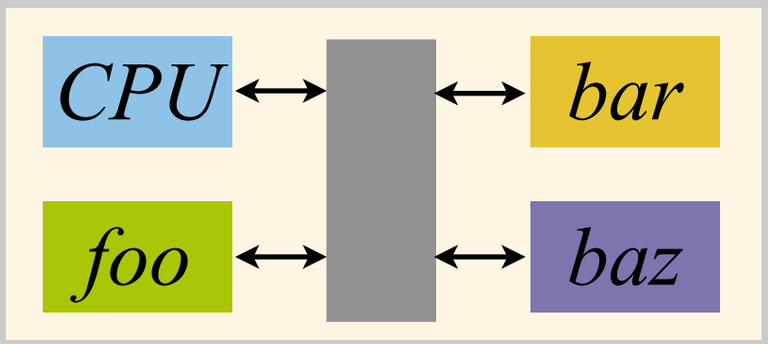




Computation Model

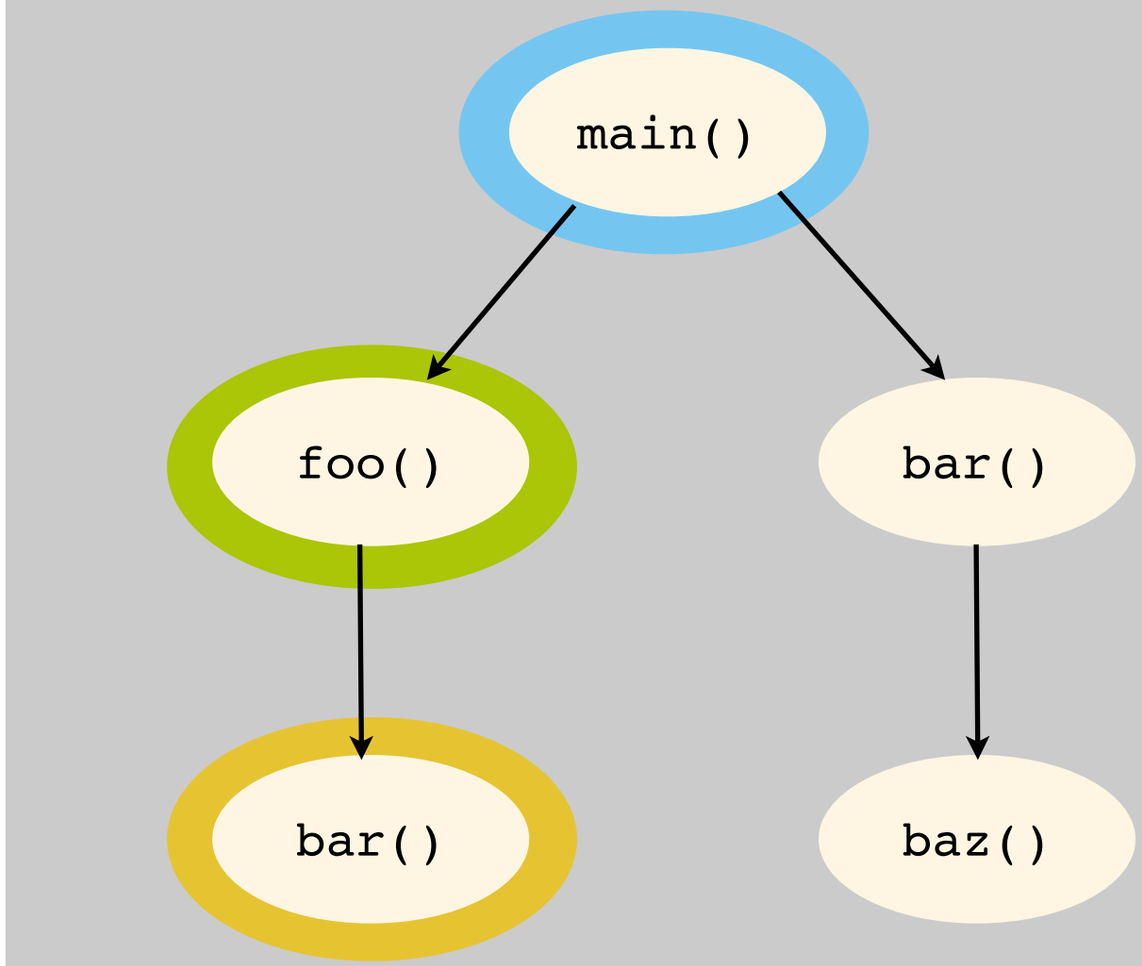
1. Dynamic function invocations mapped to corresponding accelerator

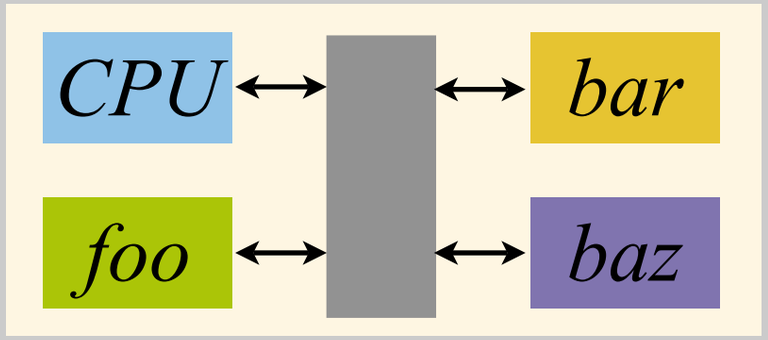




Computation Model

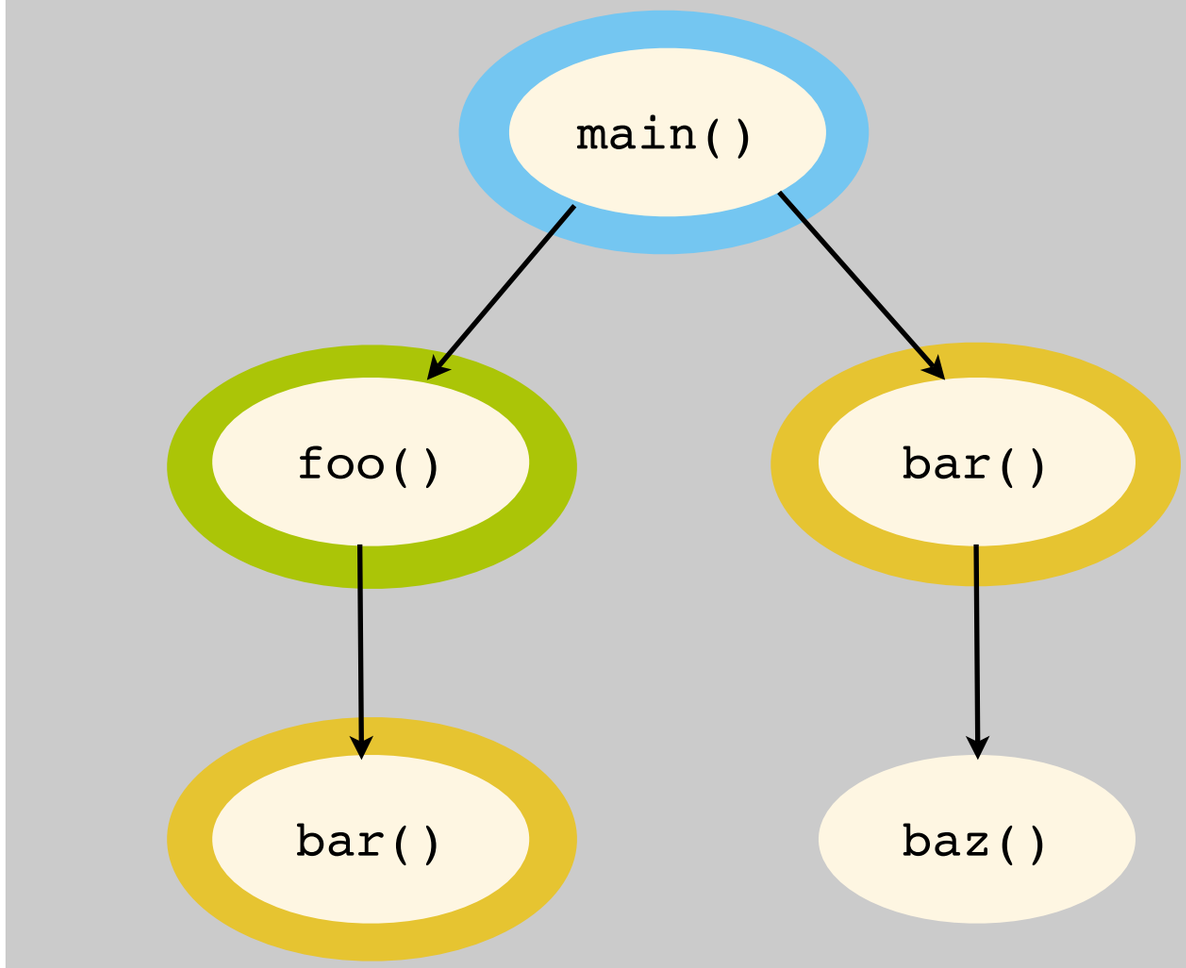
1. Dynamic function invocations mapped to corresponding accelerator

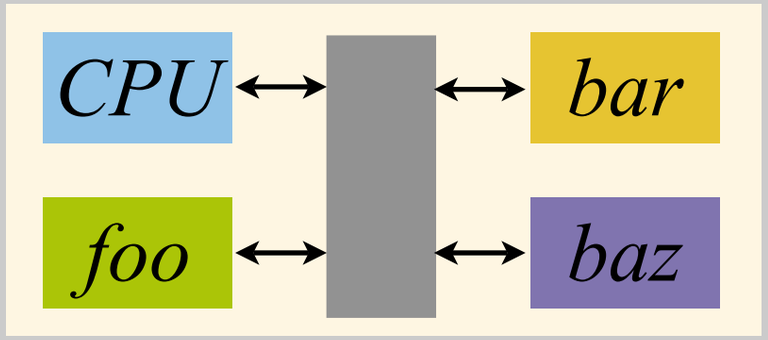




Computation Model

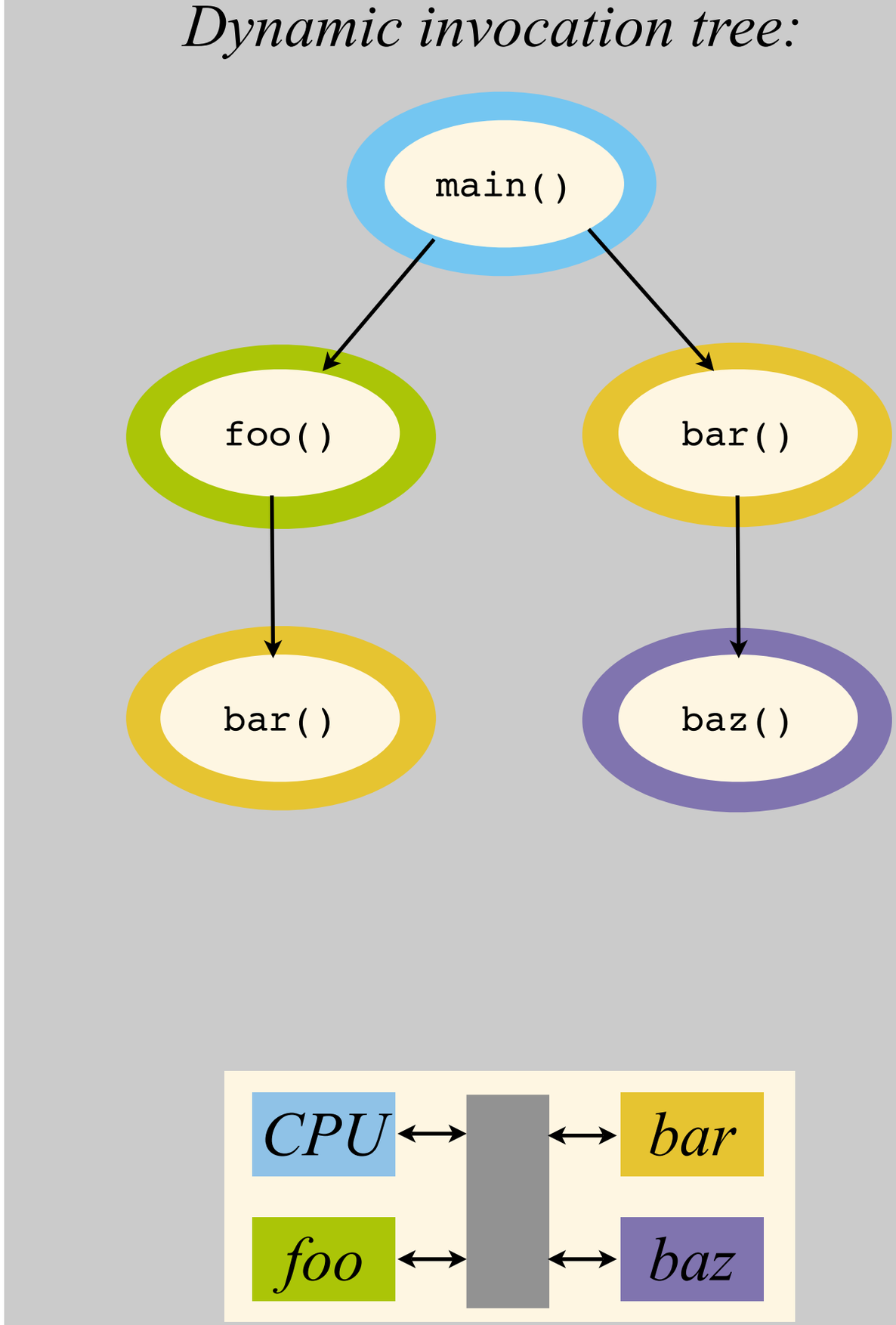
1. Dynamic function invocations mapped to corresponding accelerator





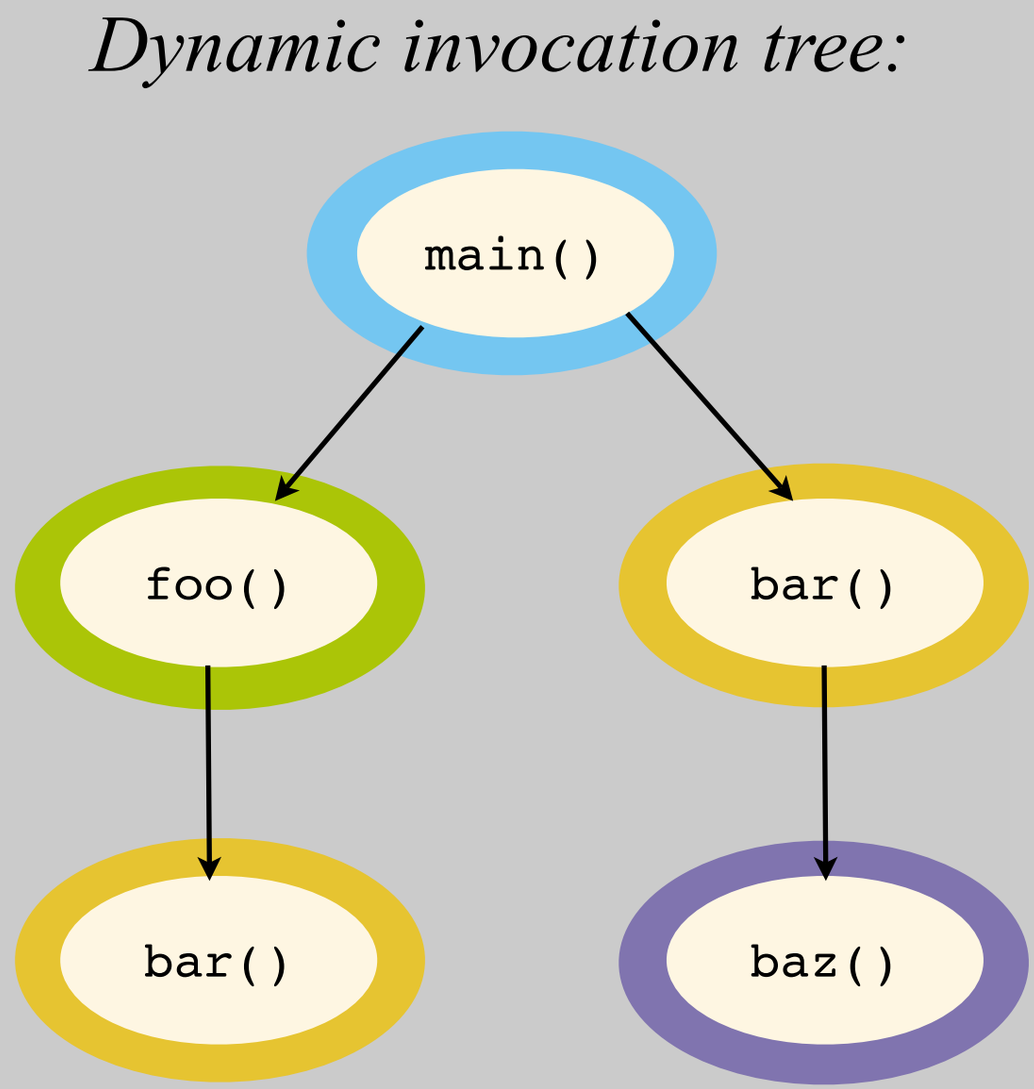
Computation Model

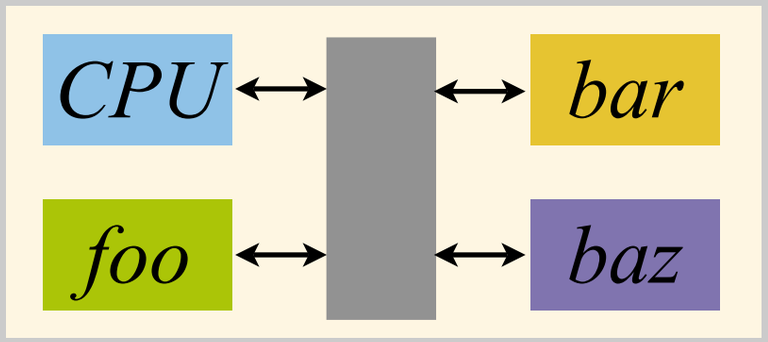
1. Dynamic function invocations mapped to corresponding accelerator



Computation Model

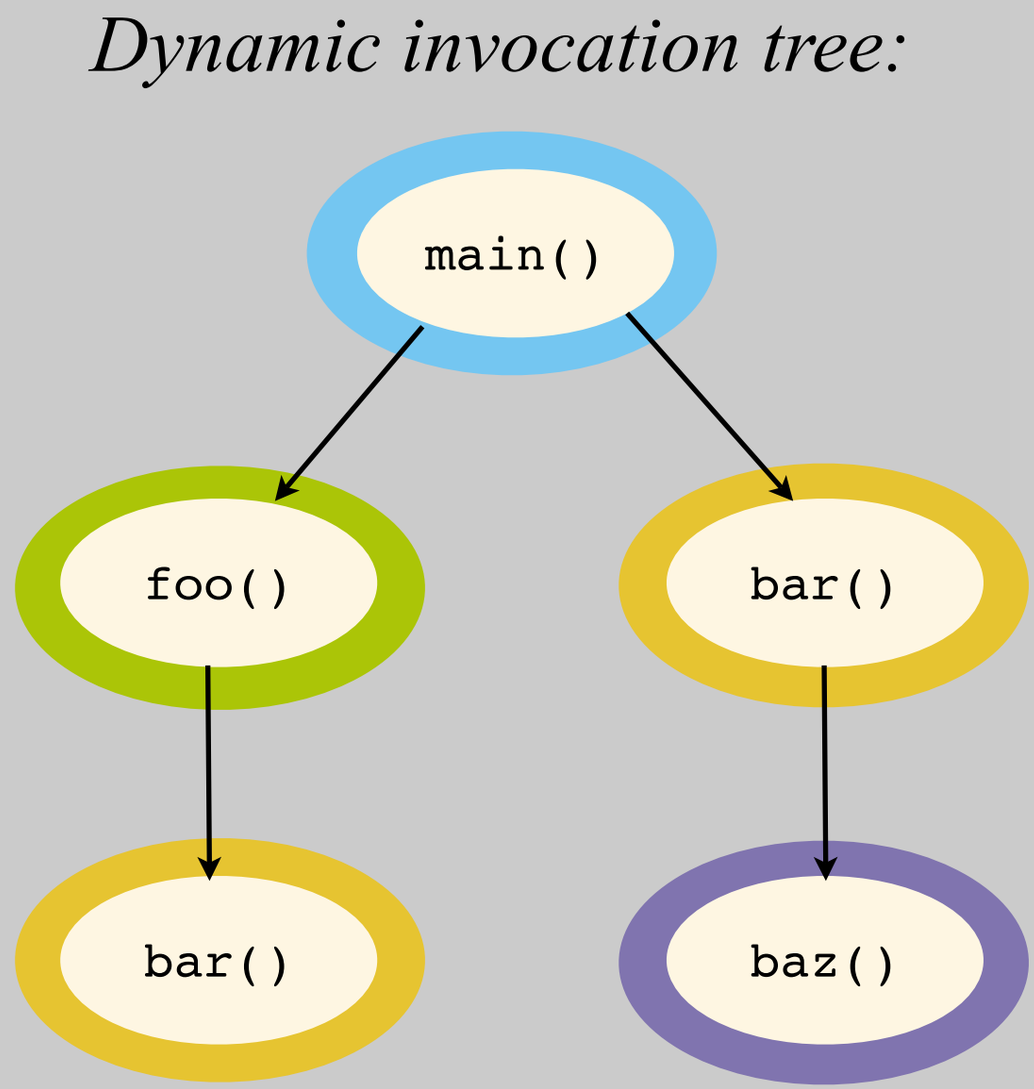
1. Dynamic function invocations mapped to corresponding accelerator
2. Invocations without accelerator are executed on same core as parent

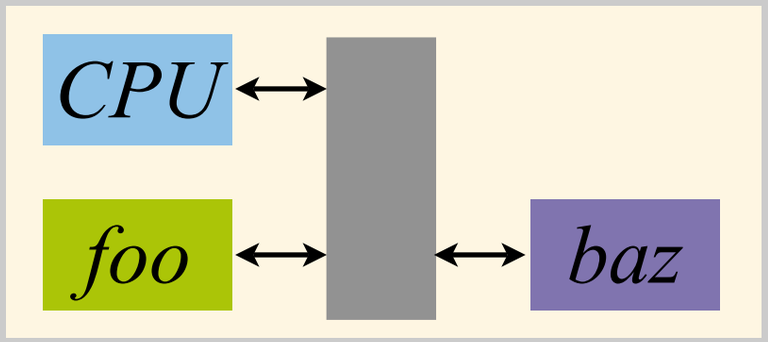




Computation Model

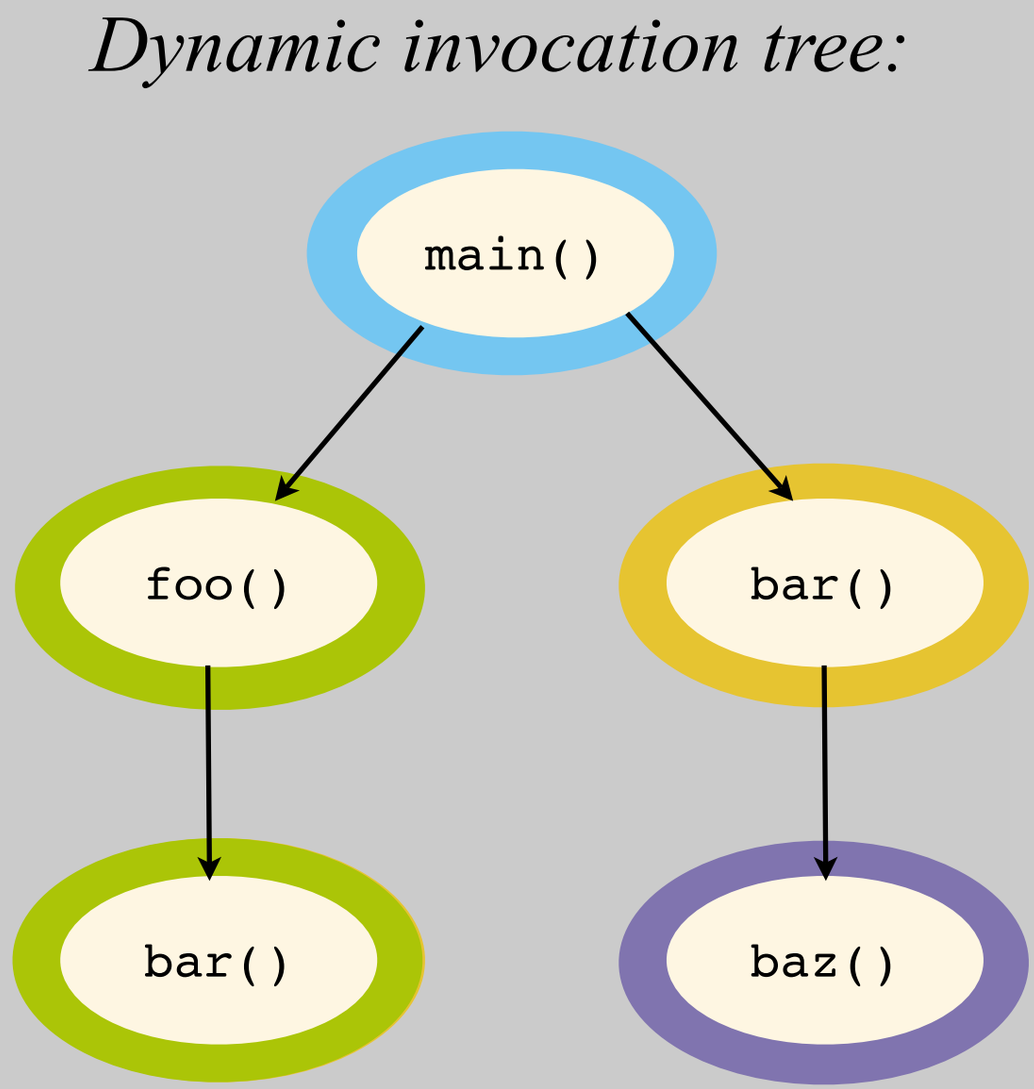
1. Dynamic function invocations mapped to corresponding accelerator
2. Invocations without accelerator are executed on same core as parent

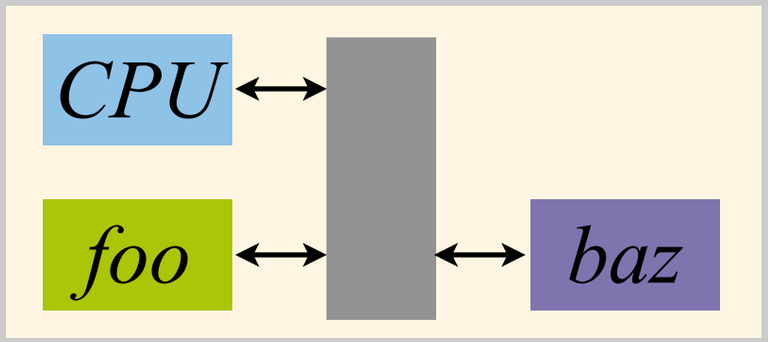




Computation Model

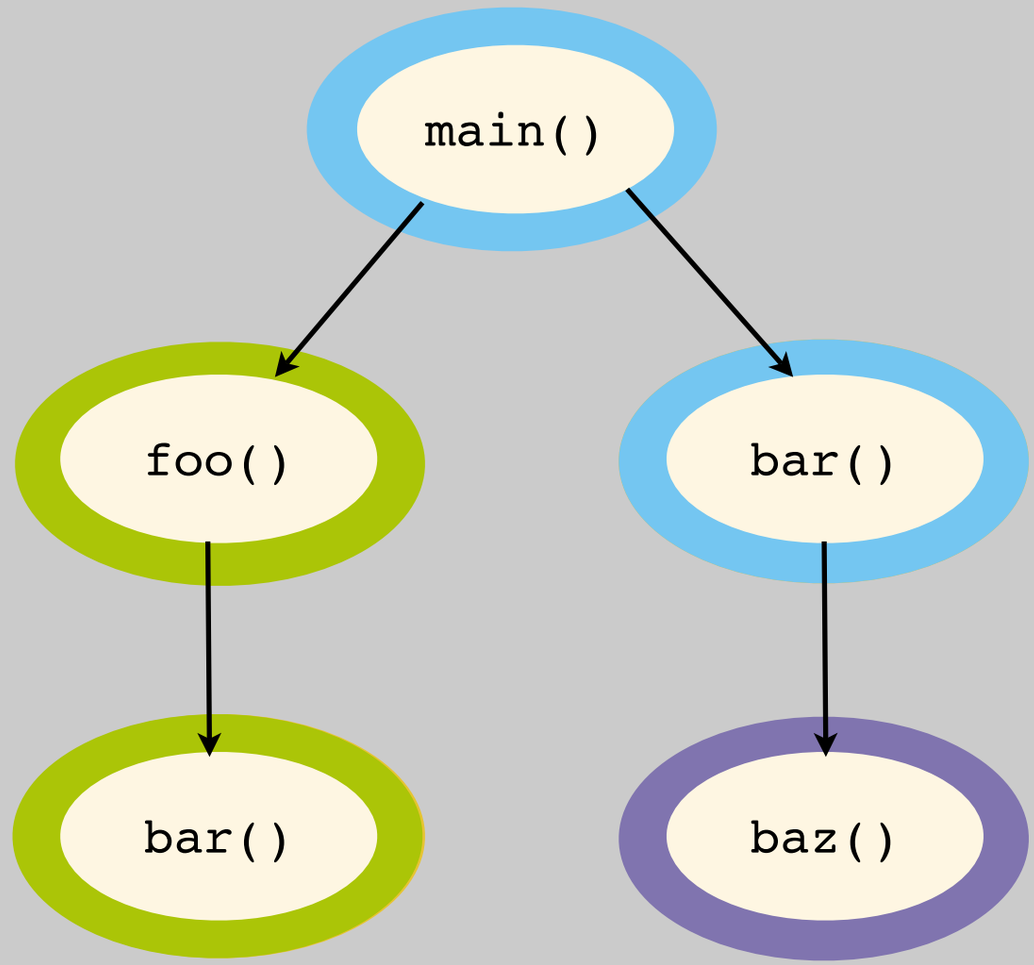
1. Dynamic function invocations mapped to corresponding accelerator
2. Invocations without accelerator are executed on same core as parent

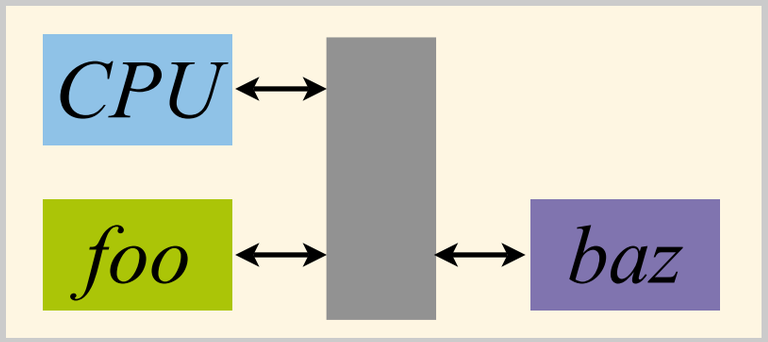




Computation Model

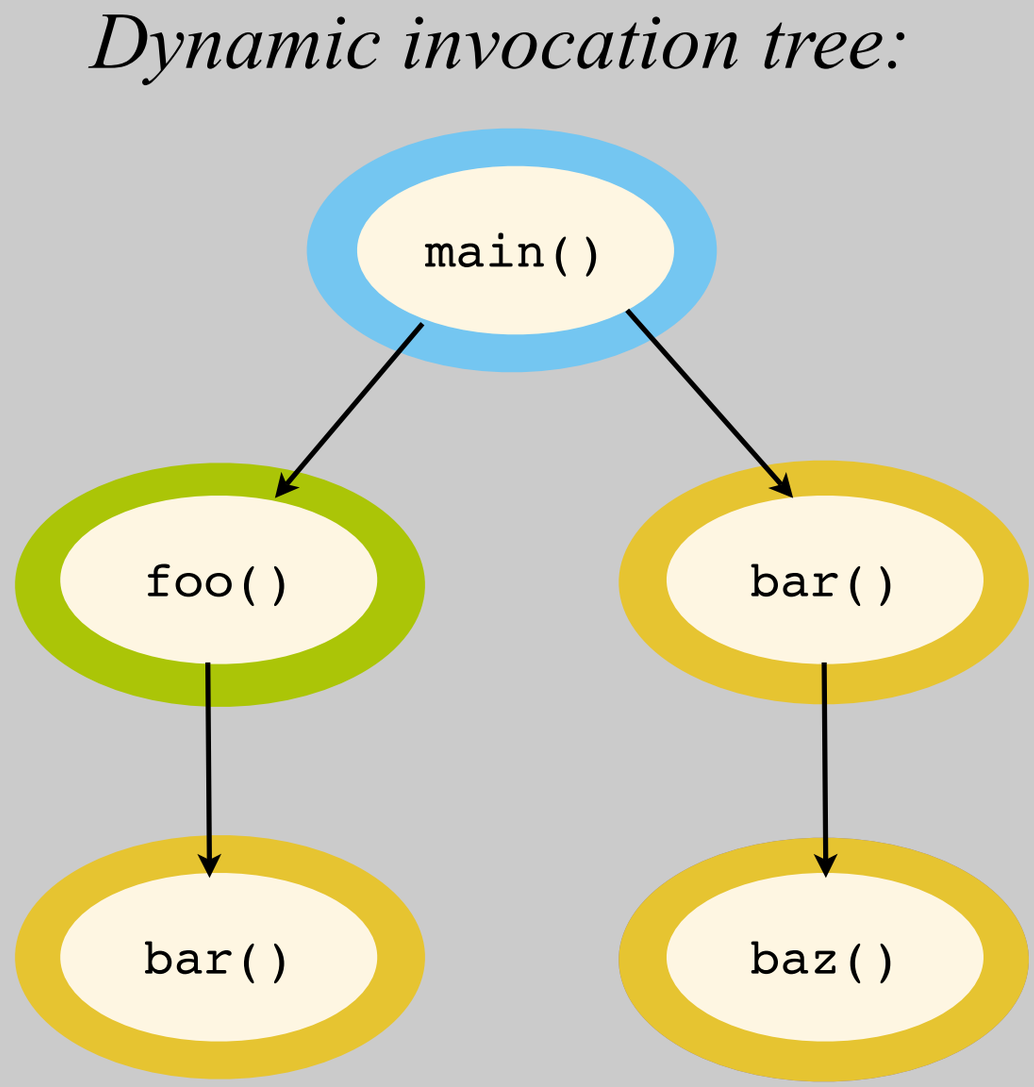
1. Dynamic function invocations mapped to corresponding accelerator
2. Invocations without accelerator are executed on same core as parent

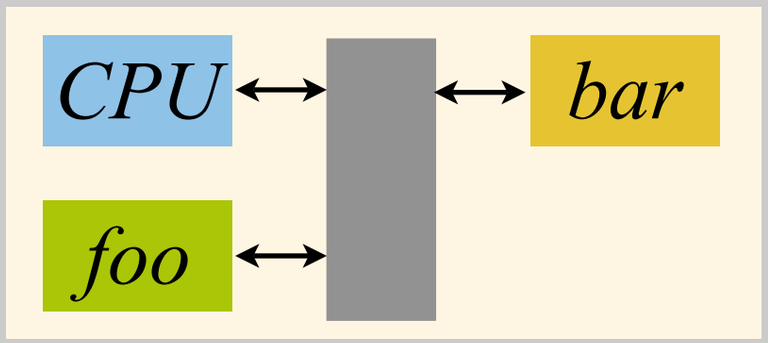




Computation Model

1. Dynamic function invocations mapped to corresponding accelerator
2. Invocations without accelerator are executed on same core as parent

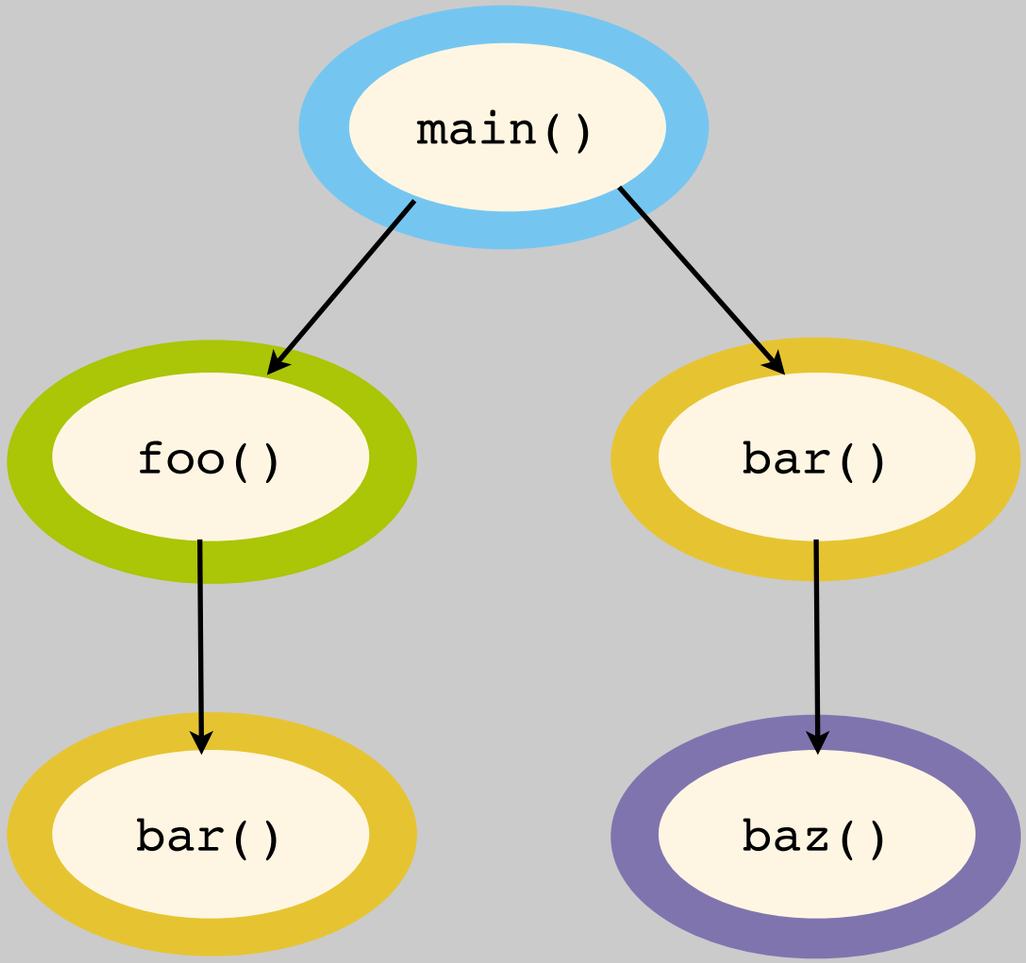


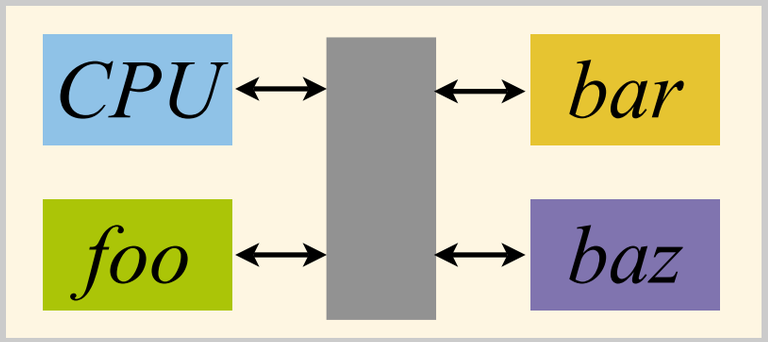


Communication Model

Invocations communicate via load/ store dependencies

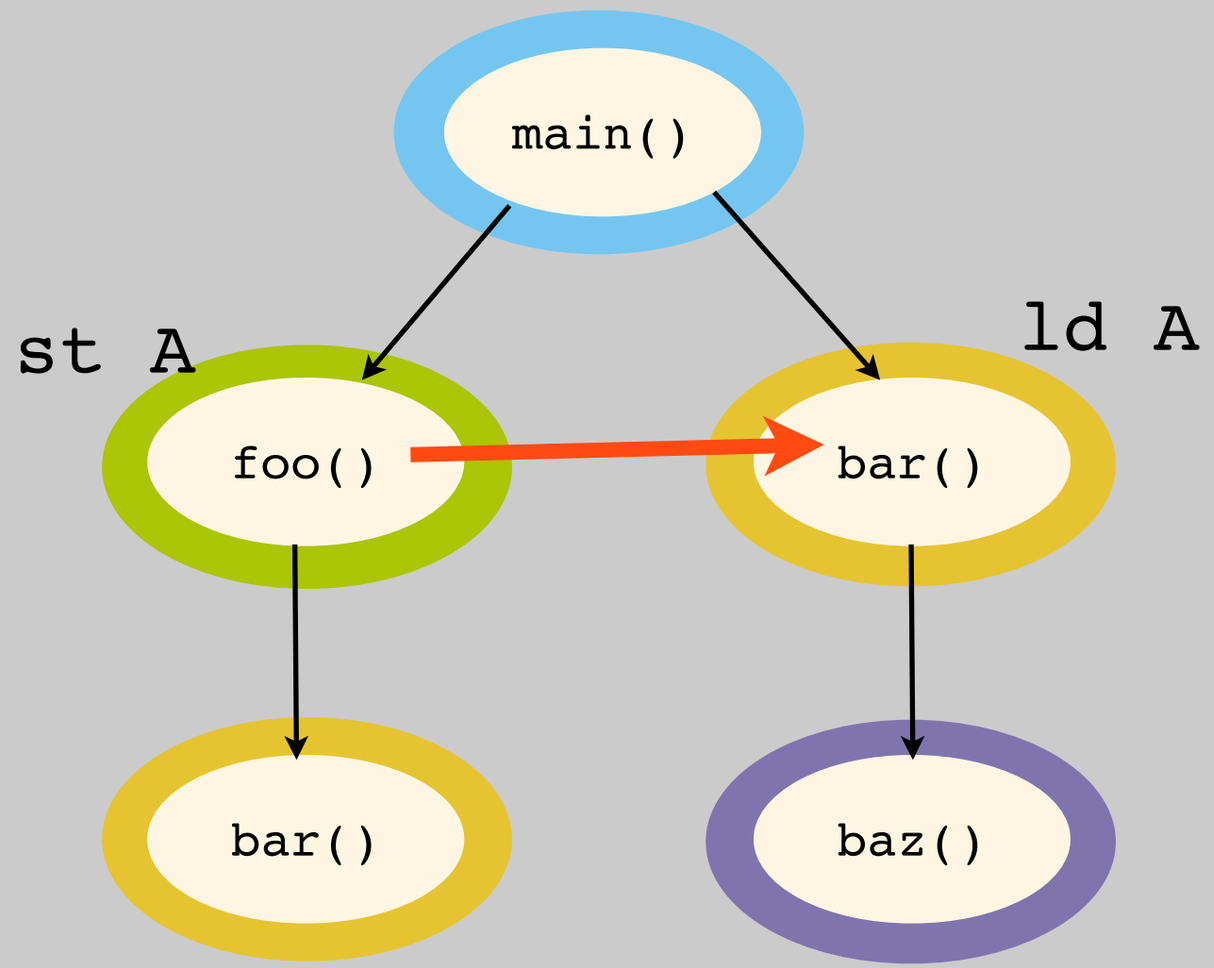
Dynamic invocation tree:

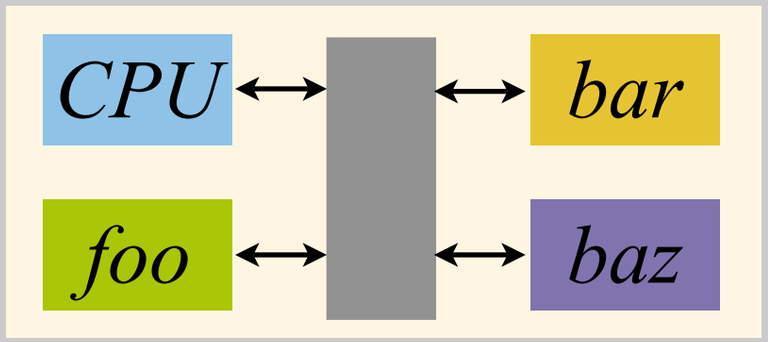




Communication Model

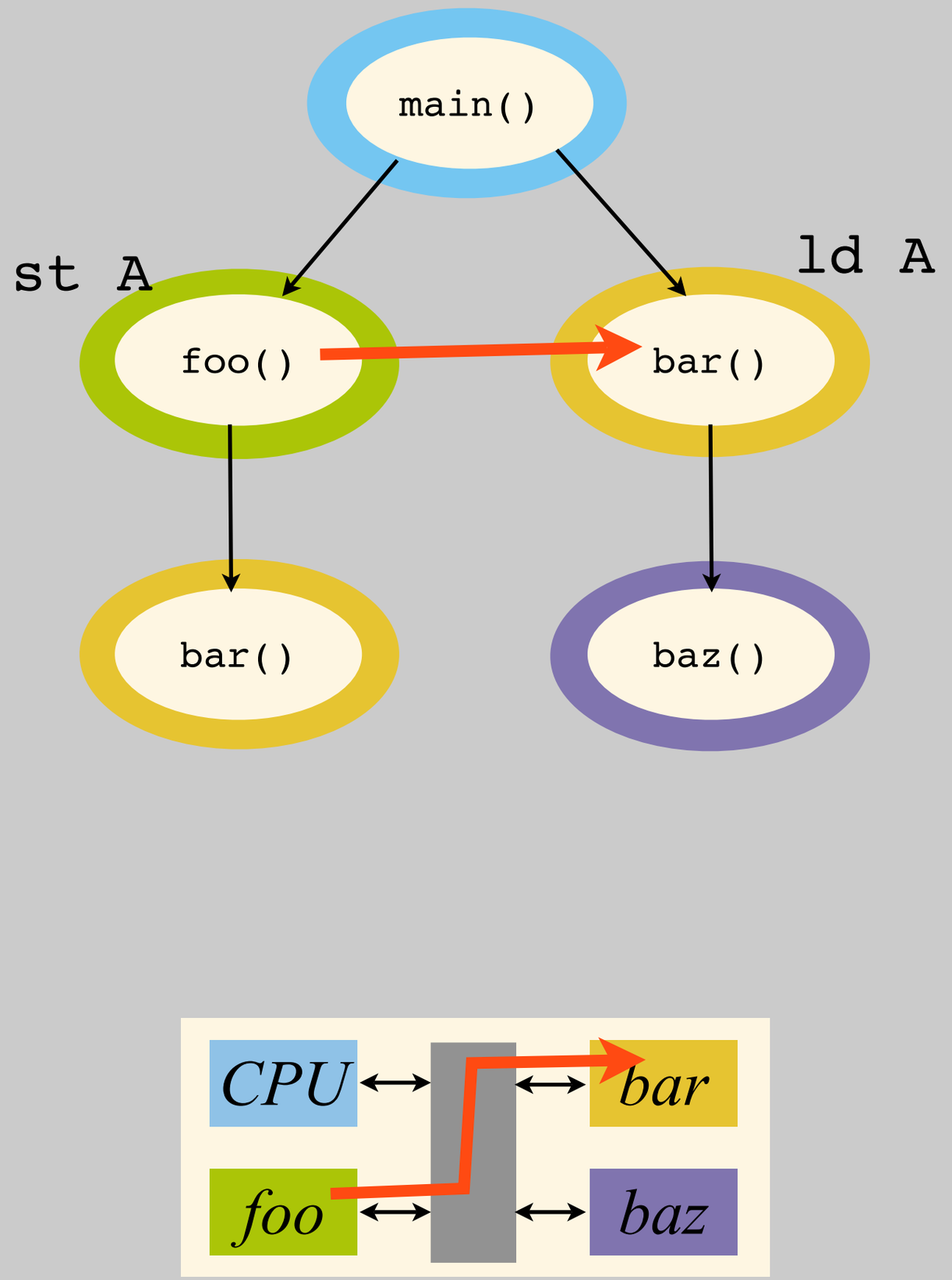
Invocations communicate via load/ store dependencies





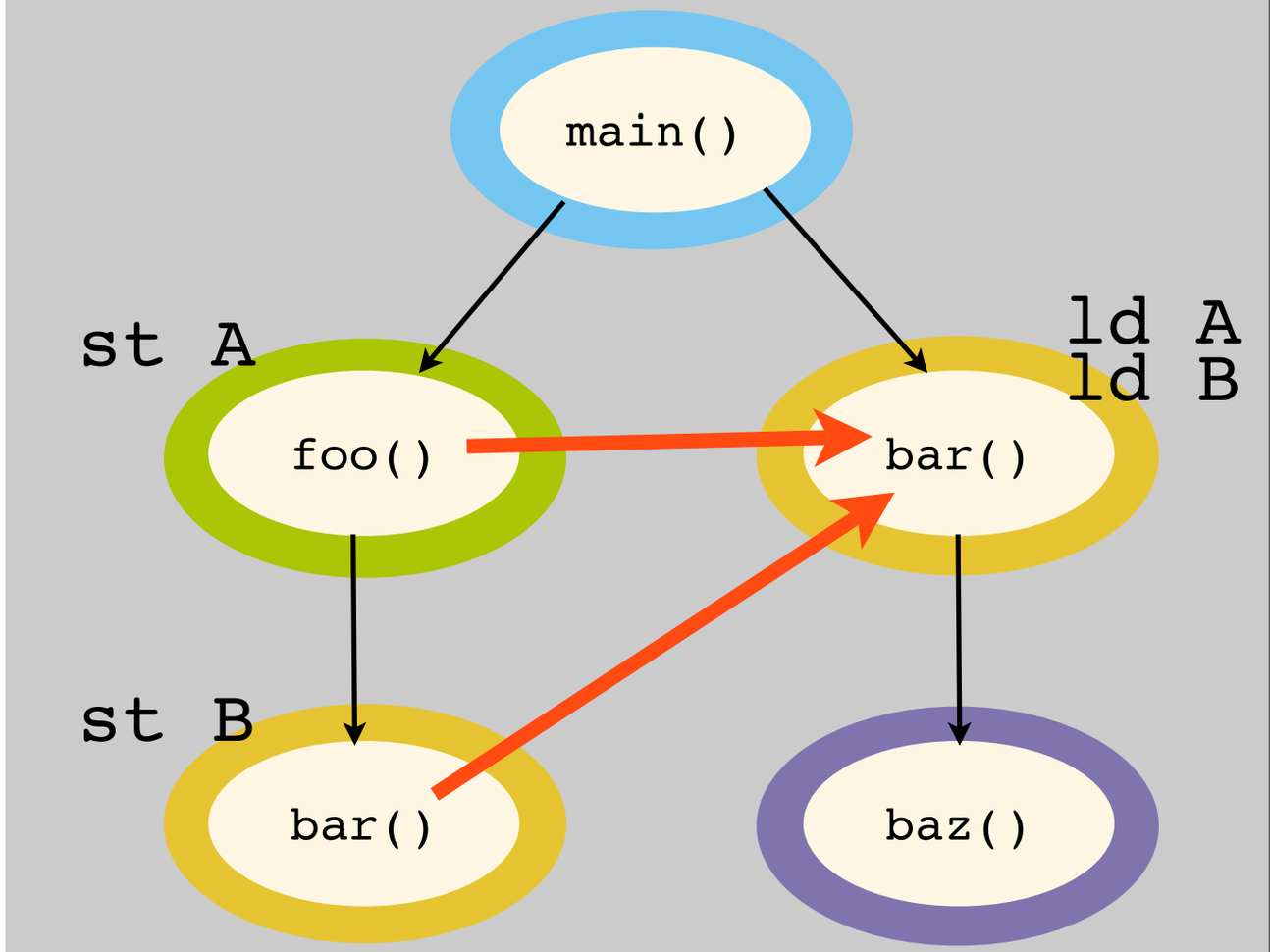
Communication Model

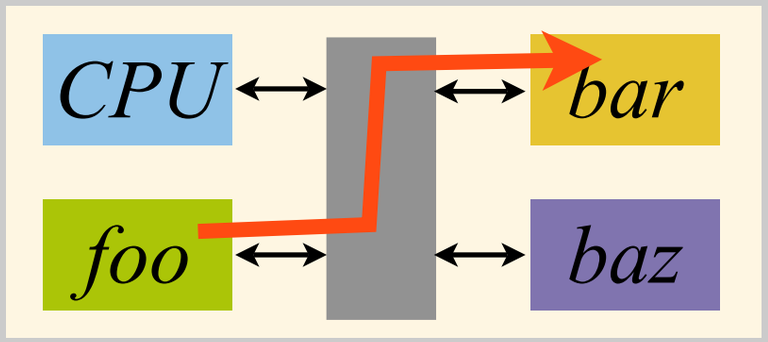
Invocations communicate via load/ store dependencies



Communication Model

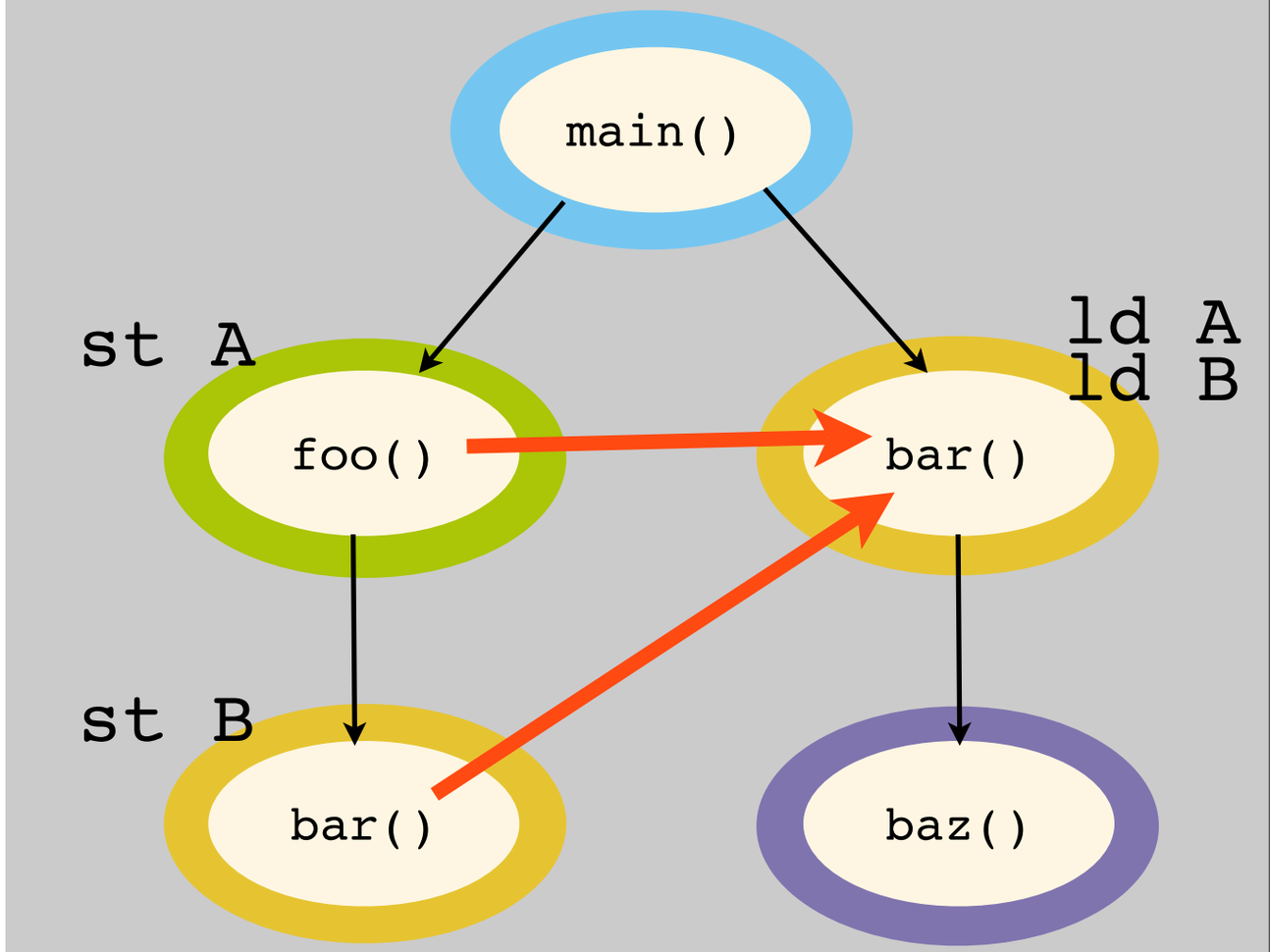
Invocations communicate via load/ store dependencies

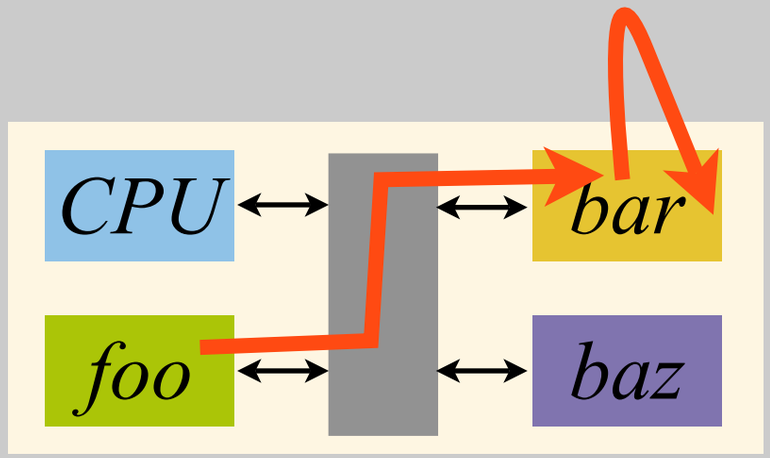




Communication Model

Invocations communicate via load/ store dependencies



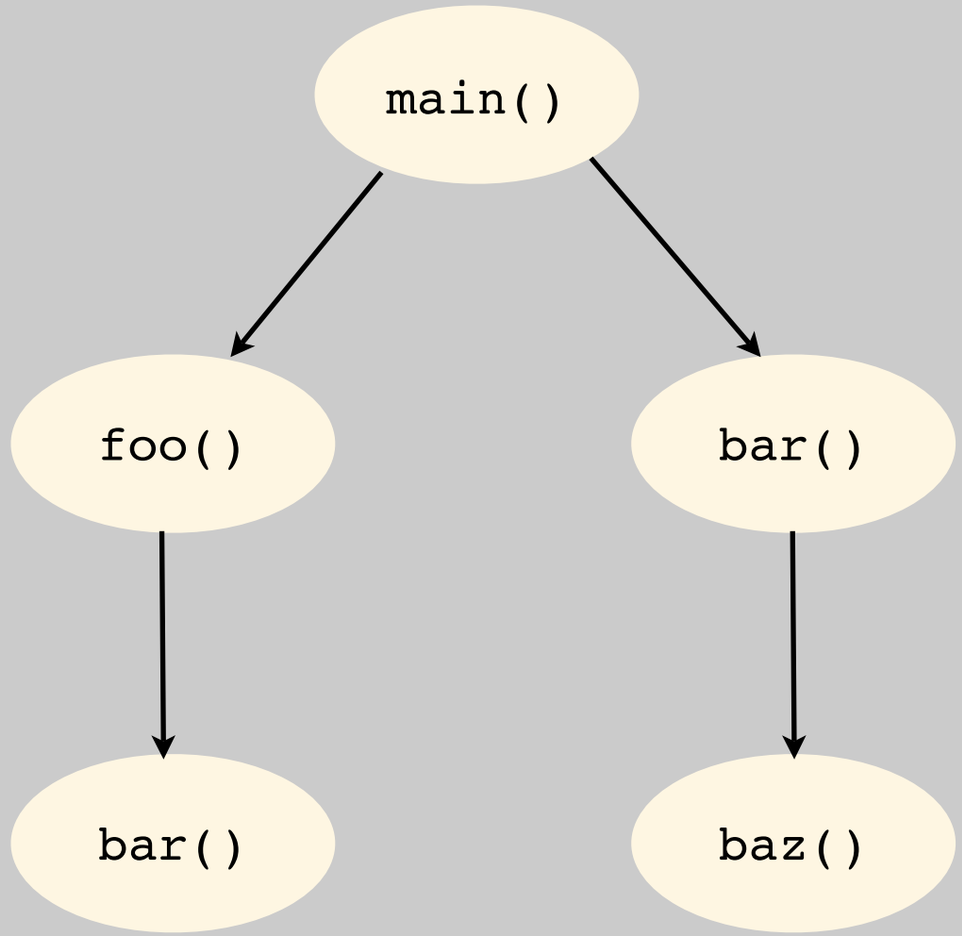


Talk Outline

- Overall Vision
- Accelerator System Model
- Methodology Overview
- Methodology in Practice
  - Image Rotation
  - JPEG
- Conclusion

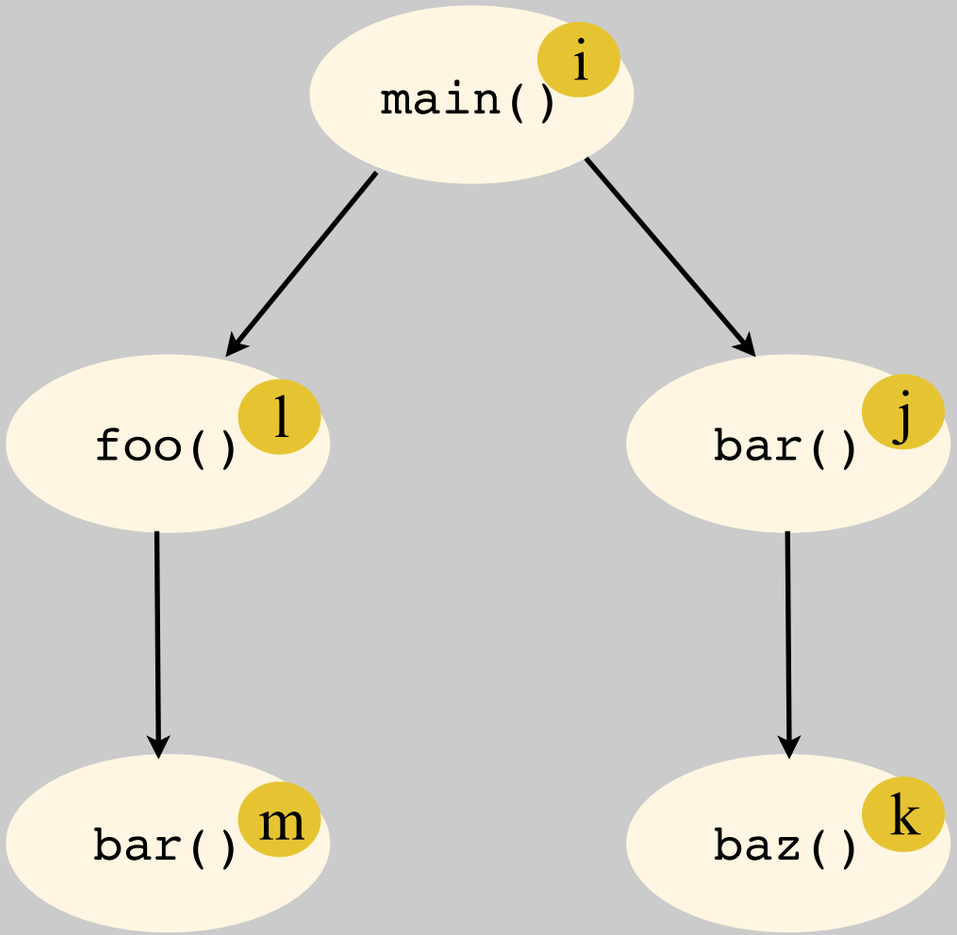
Methodology Pt 1: *Examine Application*

Pintool ➔ decorated call graph



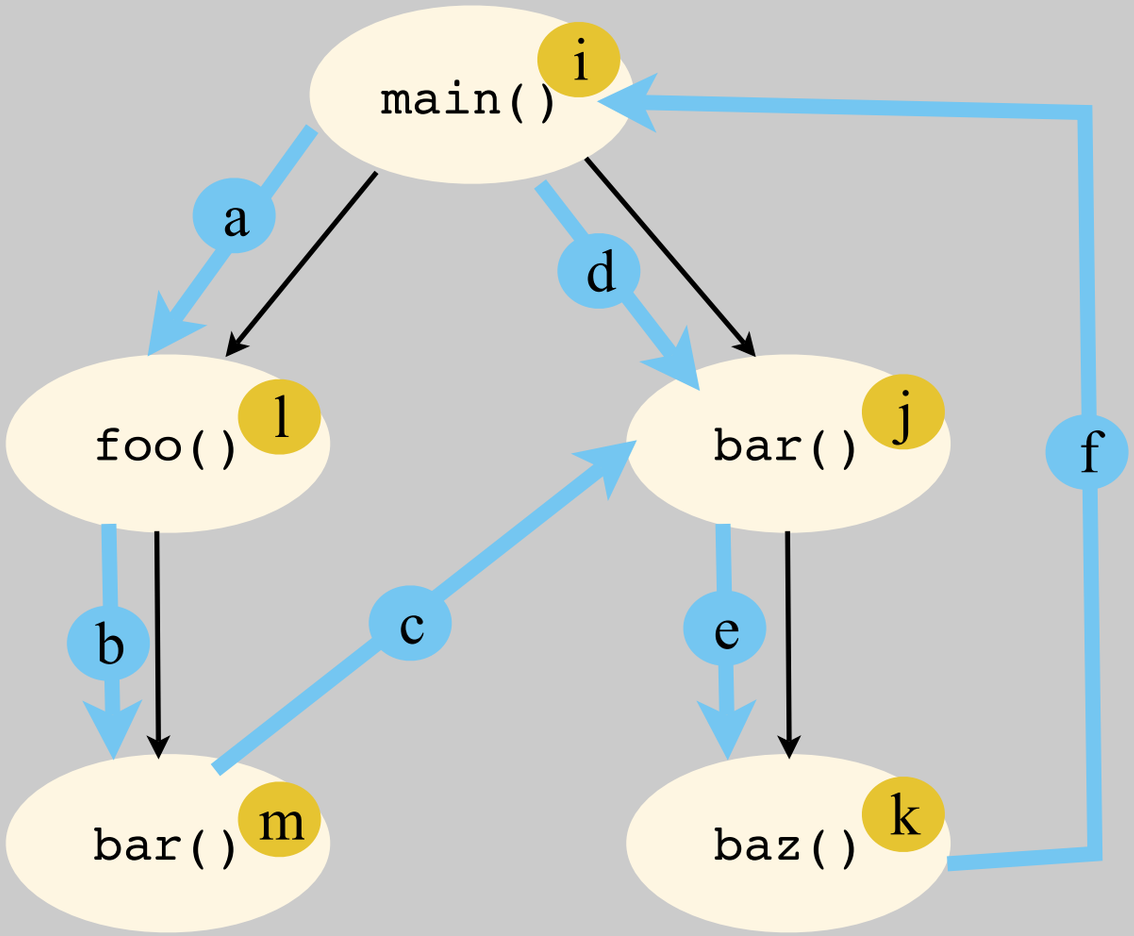
Methodology Pt 1: *Examine Application*

Pintool ➔ decorated call graph



Methodology Pt 1: *Examine Application*

Pintool ➔ decorated call graph



Pintool Functionality

- Tool instruments calls, returns, loads and stores

- Four logs are generated, all keyed off of a unique, invocation identifier

Function	invocation ID → function name	`(-tfunction, -bfunction)`
Subcalls	invocation ID → list of subcall IDs	`(-tsubcalls, -bsubcalls)`
Instruction Count	invocation ID → dynamic instruction count	`(-ticount, -bicount)`
Data Transfers	invocation ID → invocation ID, bytes	`(-txfers, -bxfers, -xfer-chunk`

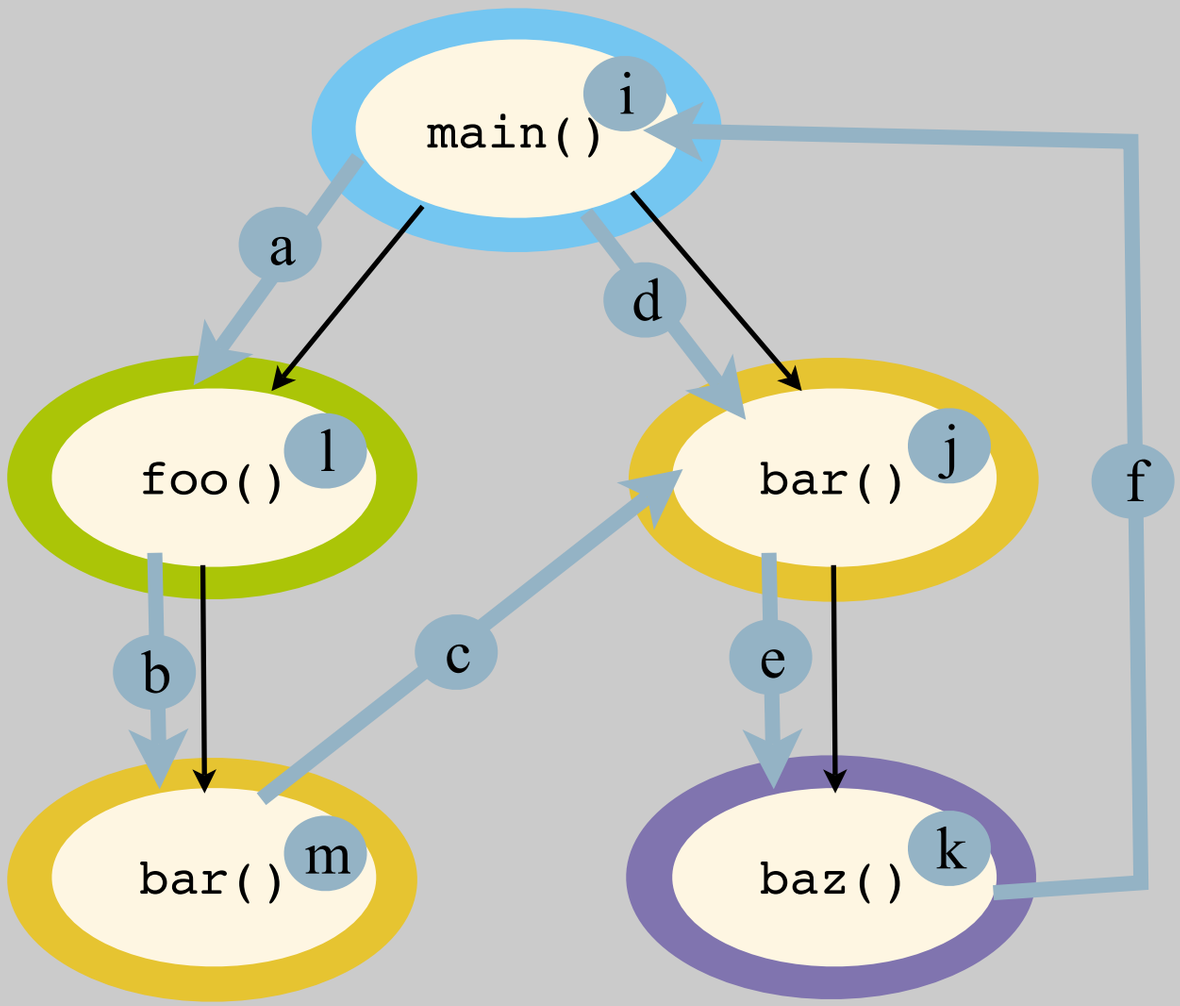
- At present, significant overhead relative to native (approximately 2000x on the short-running applications to be presented here).

- Majority of overhead attributable to hash lookups (for tracking data transfers) and logfile writing

Methodology Pt 2: *Evaluate Execution on Accelerators*

Program runtime a function of
- computation rates
- computation
- communication rates
- communication

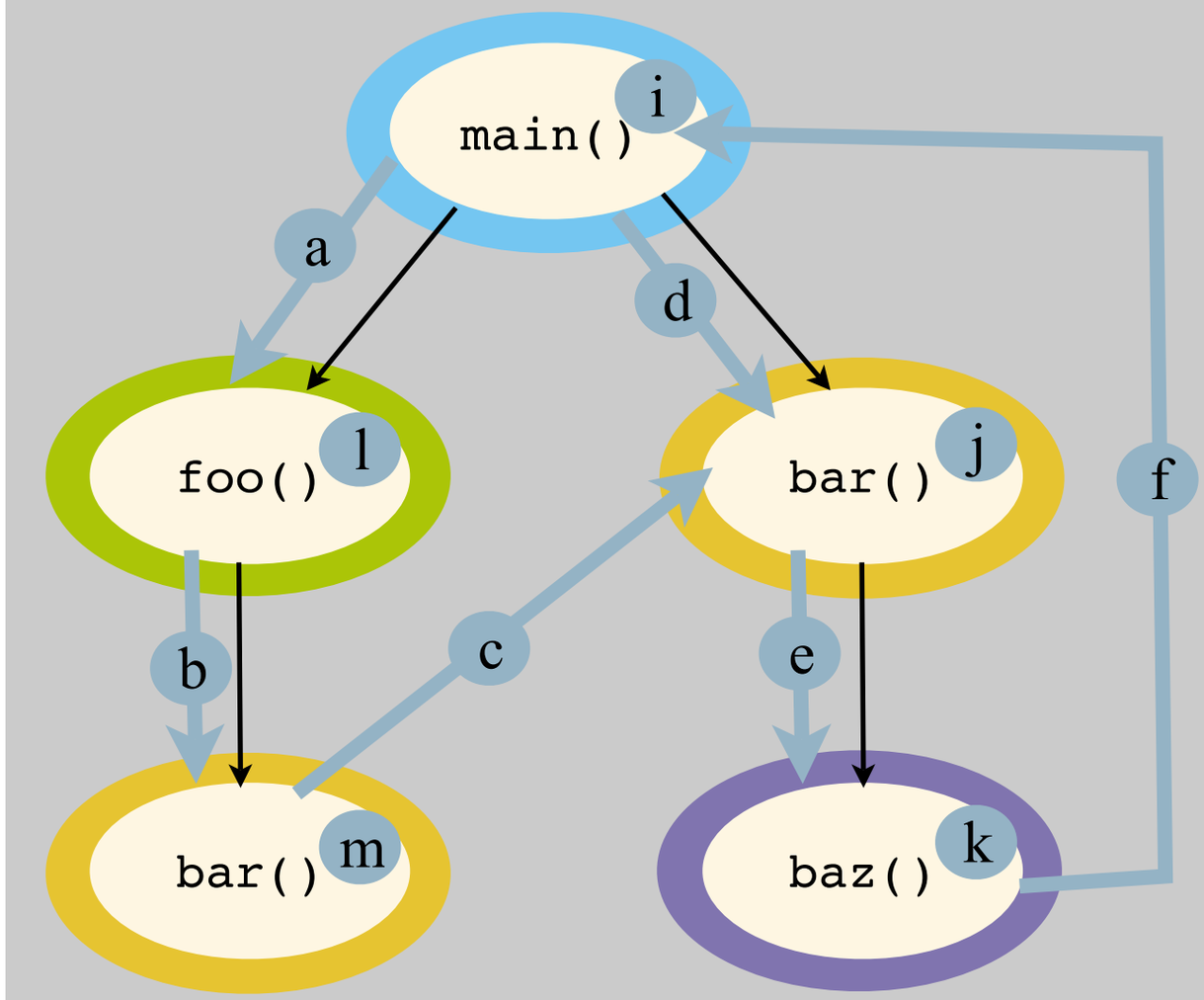
Enables evaluation of:
- acceleration potential in the limit
- sensitivity of potential to hardware parameters, program inputs, etc.



Why is gprof not sufficient?

Runtimes are machine- and algorithm- dependent

Does not capture data transfers between invocations



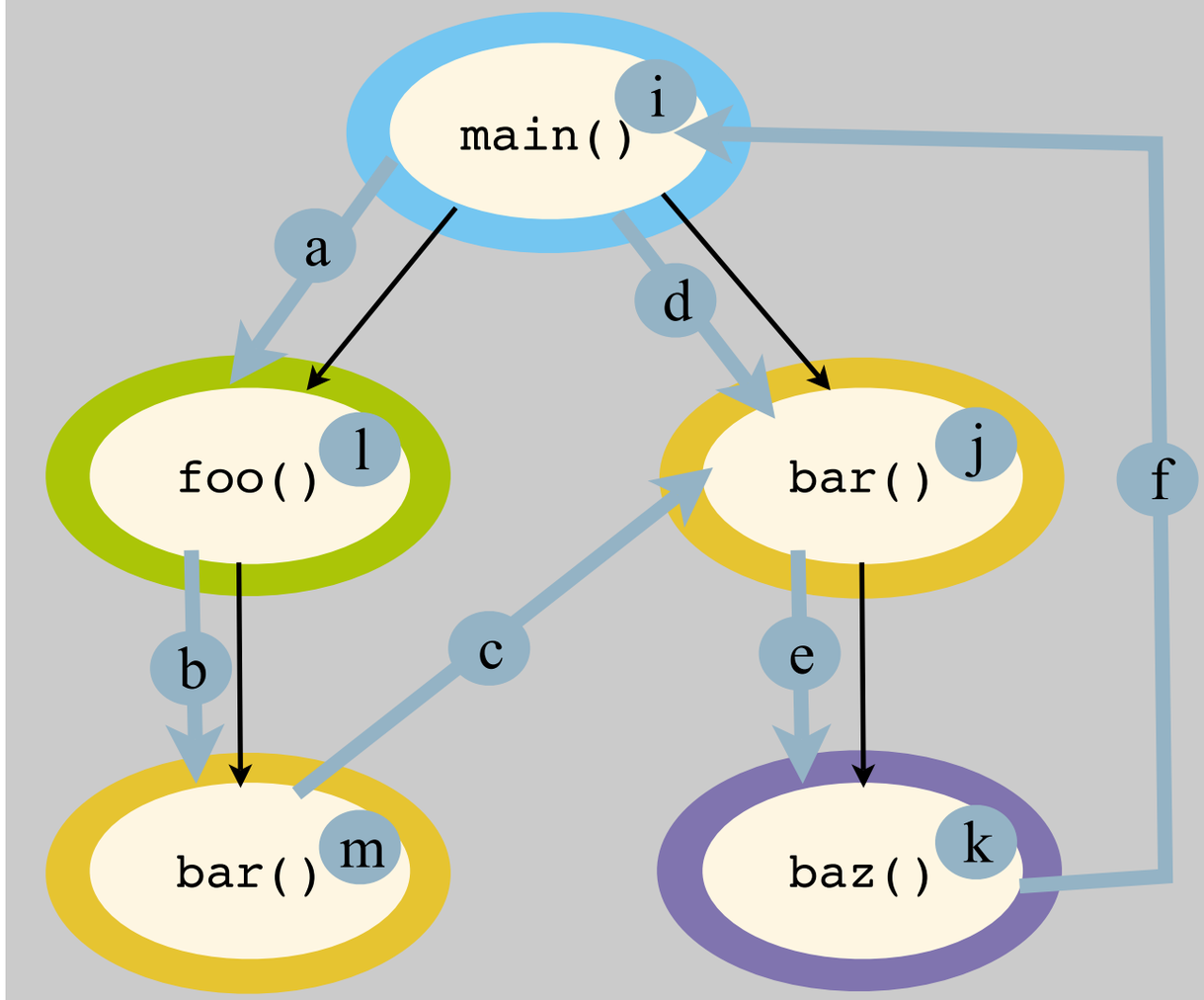
Talk Outline

- Overall Vision
- Accelerator System Model
- Methodology Overview
- Methodology in Practice
  - Image Rotation
  - JPEG
- Conclusion

Methodology In Practice

Image rotation

JPEG decode



Simple Example: Image Rotation

```
#define PIX(x,y) raster[(x) + (y)*wd]
unsigned wd, ht, maxval, *raster;

int main(int argc, char** argv) {
  if (argc != 2 || (argv[1][0] != 'r' && argv[1][0] != 'i')) {
    printf("USAGE: rotate [ir]\n"), exit(0);
  }
  read_ppm();
  if (argv[1][0] == 'r') rec_rot(0, 0, wd);
  else  iter_rot();
  write_ppm();
  return 0;
}
```

Image Rotation: Recursive Algorithm

```
#define PIX(x,y) raster[(x) + (y)*wd]
unsigned wd, ht, maxval, *raster;

int main(int argc, char** argv) {
    if (argc != 2 || (argv[1][0] != 'r' && argv
        printf("USAGE: rotate [ir]\n"), exit(0)
    }
    read_ppm();
    if (argv[1][0] == 'r') rec_rot(0, 0, wd);
    else   iter_rot();
    write_ppm();
    return 0;
}
```

```
void rec_rot(int x, int y, int s) {
  int i, j;
  s >>= 1;
  for (i = 0 ; i < s ; ++i)
    for (j = 0 ; j < s ; ++j) {
      int rgb = PIX(x+i, y+j);
      PIX(x+i,   y+j  ) = PIX(x+i,   y+j+s);
      PIX(x+i,   y+j+s) = PIX(x+i+s, y+j+s);
      PIX(x+i+s, y+j+s) = PIX(x+i+s, y+j);
      PIX(x+i+s, y+j  ) = rgb;
    }
  if (s <= 1) return;

  rec_rot(x,y+s,s);
  rec_rot(x+s,y+s,s);
  rec_rot(x+s,y,s);
  rec_rot(x,y,s);
}
```

Image Rotation: Iterative Algorithm

```
#define PIX(x,y) raster[(x) + (y)*wd]
unsigned wd, ht, maxval, *raster;

int main(int argc, char** argv) {
    if (argc != 2 || (argv[1][0] != 'r' && argv[1][0] != 'i')) {
        printf("USAGE: rotate [ir]\n"), exit(0);
    }
    read_ppm();
    if (argv[1][0] == 'r') rec_rot(0, 0, wd);
    else iter_rot();
    write_ppm();
    return 0;
}
```

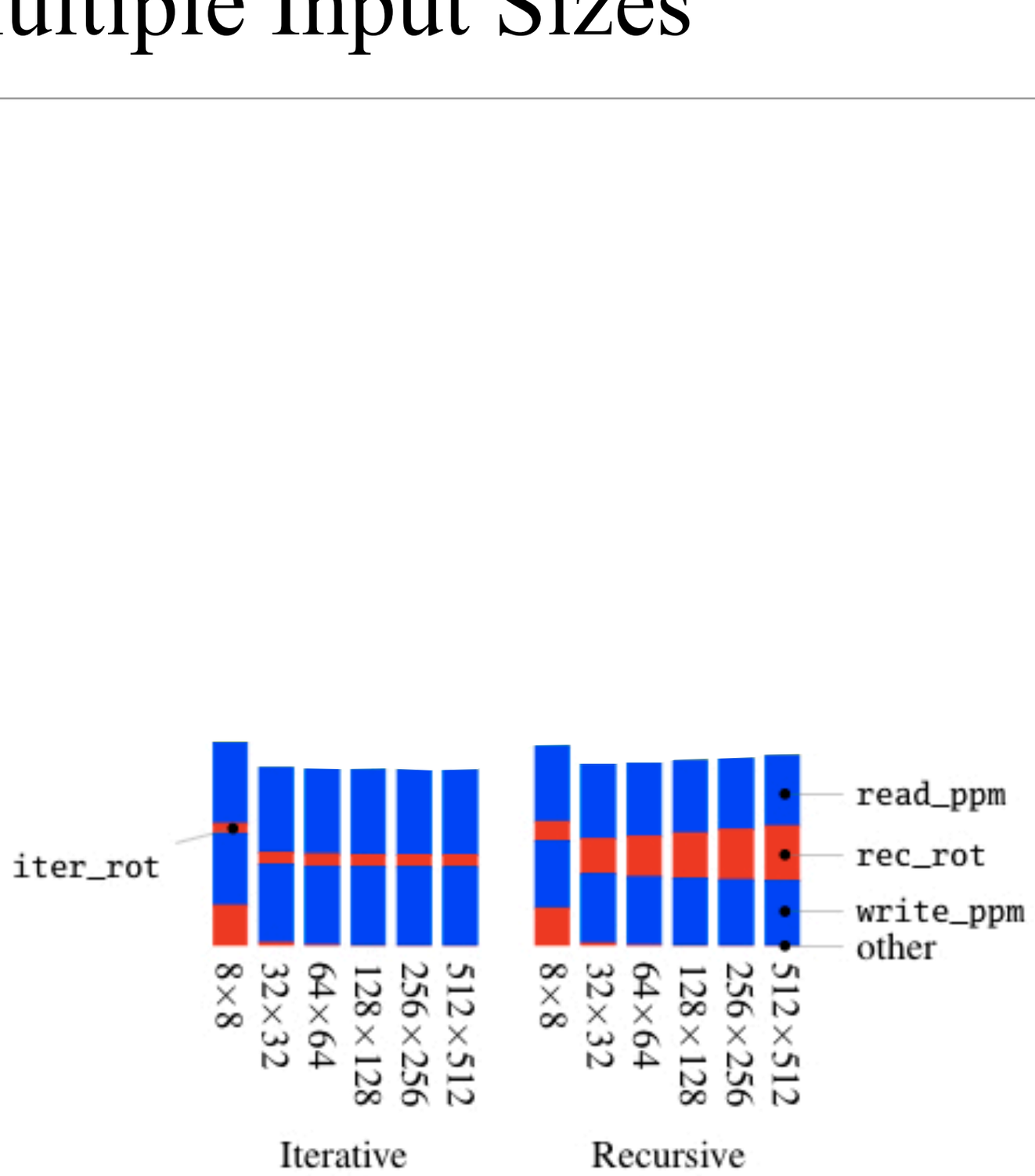
```
void iter_rot() {
  int x, y, s;
  s = wd >> 1;
  for (y = 0 ; y < s ; ++y)
    for (x = 0 ; x < s ; ++x) {
      int rgb = PIX(x, y);
      PIX(x,       y       ) = PIX(y,       ht-x-1);
      PIX(y,       ht-x-1) = PIX(wd-x-1, ht-y-1);
      PIX(wd-x-1, ht-y-1) = PIX(wd-y-1, x      );
      PIX(wd-y-1, x       ) = rgb;
    }
}
```

Experiment 1: Multiple Input Sizes

- Relative ratio of computation and communication nearly constant across **algorithms** and **input** sizes

- Useful for high level survey of input space

- Given this result, will proceed to analyze a single input image size (128x128)

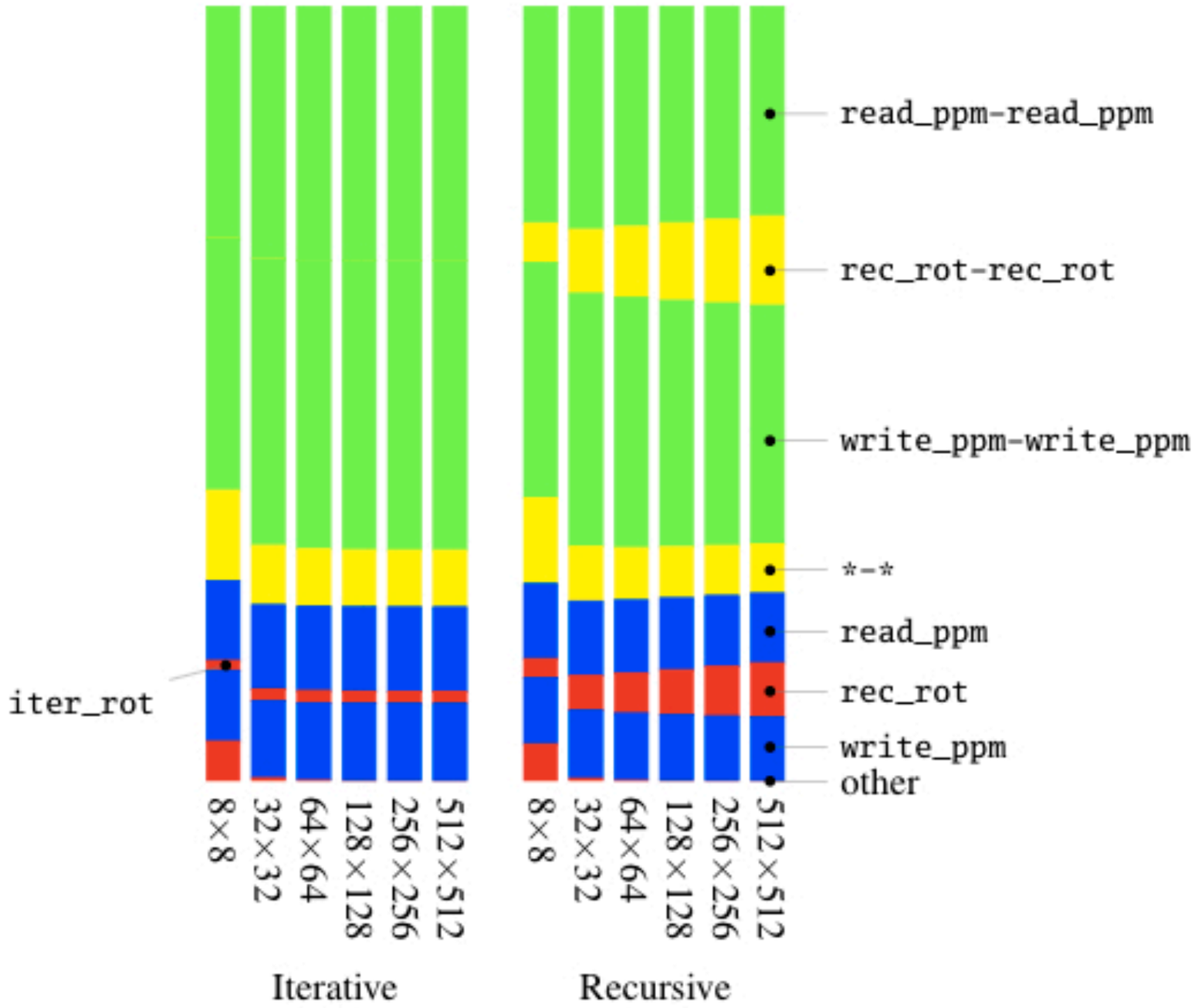


Experiment 1: Multiple Input Sizes

- Relative ratio of computation and communication nearly constant across **algorithms** and **input** sizes

- Useful for high level survey of input space

- Given this result, will proceed to analyze a single input image size (128x128)



Experiment 2: Hotspots (Iterative Rotate)

Computational and communication hotspots become visible

Communication volume (bytes)

	`read_ppm`	`iter_rot`	`write_ppm`	*other*
`read_ppm`	607413	0	0	49172
`iter_rot`	16387	4101	0	0
`write_ppm`	5	16384	689163	49155
other	4	0	18	491

Computation

804340	114820	738584	1902

Experiment 2: Hotspots (Iterative Rotate)

Computational and communication hotspots become visible

Communication volume (bytes)

	`read_ppm`	`iter_rot`	`write_ppm`	*other*
`read_ppm`	607413	0	0	49172
`iter_rot`	16387	4101	0	0
`write_ppm`	5	16384	689163	49155
other	4	0	18	491

Computation

804340	114820	738584	1902

Experiment 2: Hotspots (Iterative Rotate)

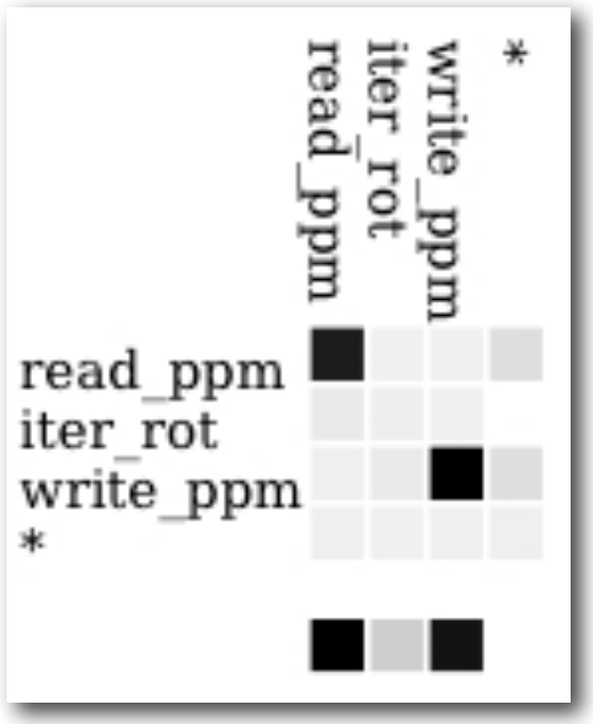
Computational and communication hotspots become visible

Communication volume (bytes)

	`read_ppm`	`iter_rot`	`write_ppm`	*other*
`read_ppm`	607413	0	0	49172
`iter_rot`	16387	4101	0	0
`write_ppm`	5	16384	689163	49155
other	4	0	18	491

Computation

804340	114820	738584	1902



Experiment 2: Hotspots (Iterative Rotate)

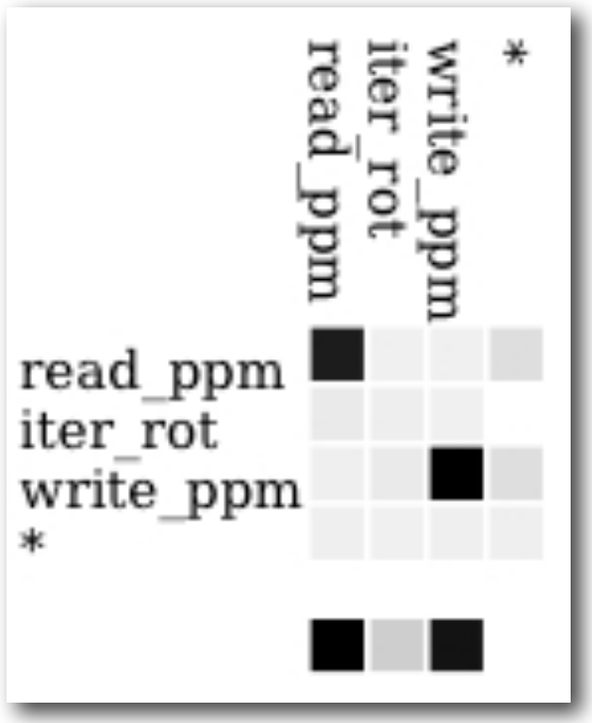
Computational and communication hotspots become visible

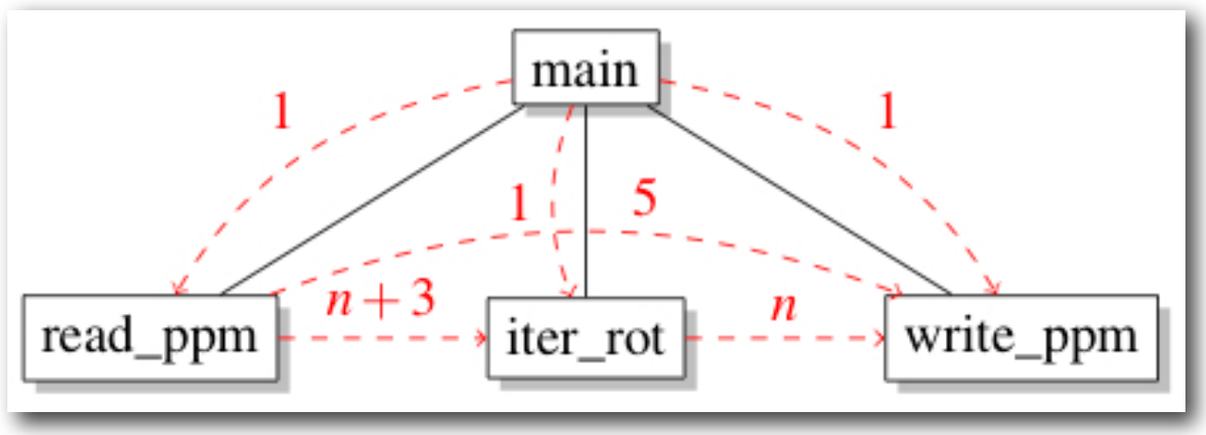
Communication volume (bytes)

	`read_ppm`	`iter_rot`	`write_ppm`	*other*
`read_ppm`	607413	0	0	49172
`iter_rot`	16387	4101	0	0
`write_ppm`	5	16384	689163	49155
other	4	0	18	491

Computation

804340	114820	738584	1902





Experiment 2: Hotspots (Recursive Rotate)

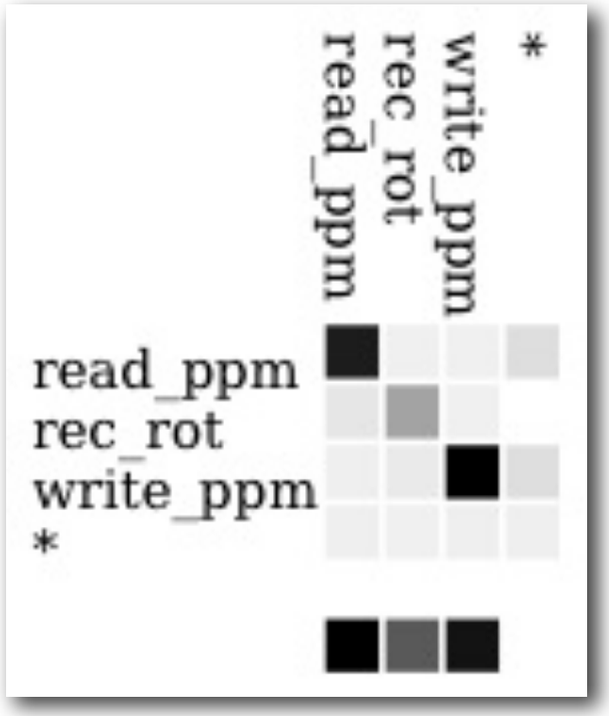
Communication volume (bytes)

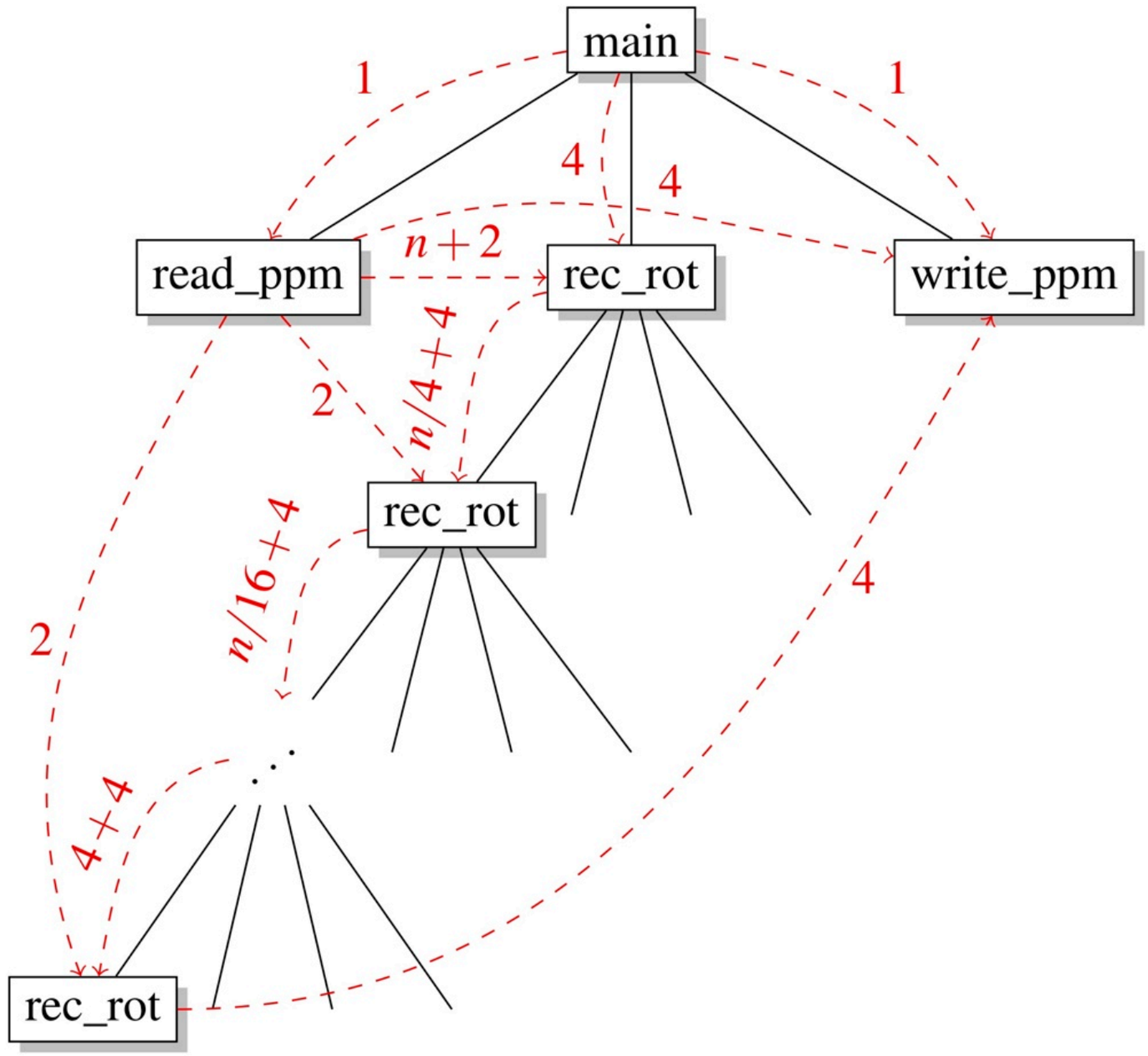
	`read_ppm`	`rec_rot`	`write_ppm`	*other*
`read_ppm`	607413	0	0	49172
`rec_rot`	27306	216888	0	4
`write_ppm`	5	16384	689163	49155
other	4	0	18	491

Computation

804340	114820	738584	1902

Experiment 2: Hotspots (Recursive Rotate)

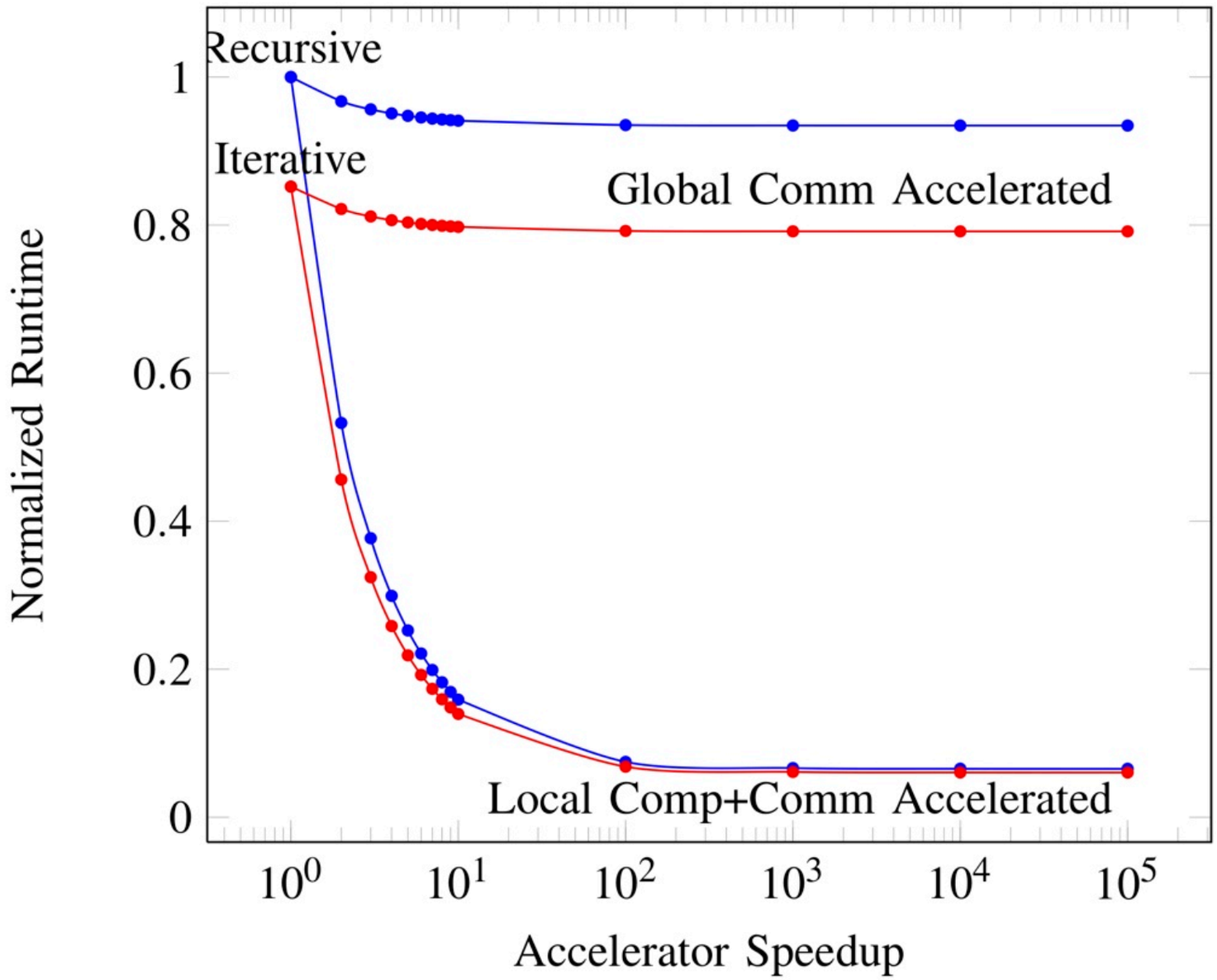




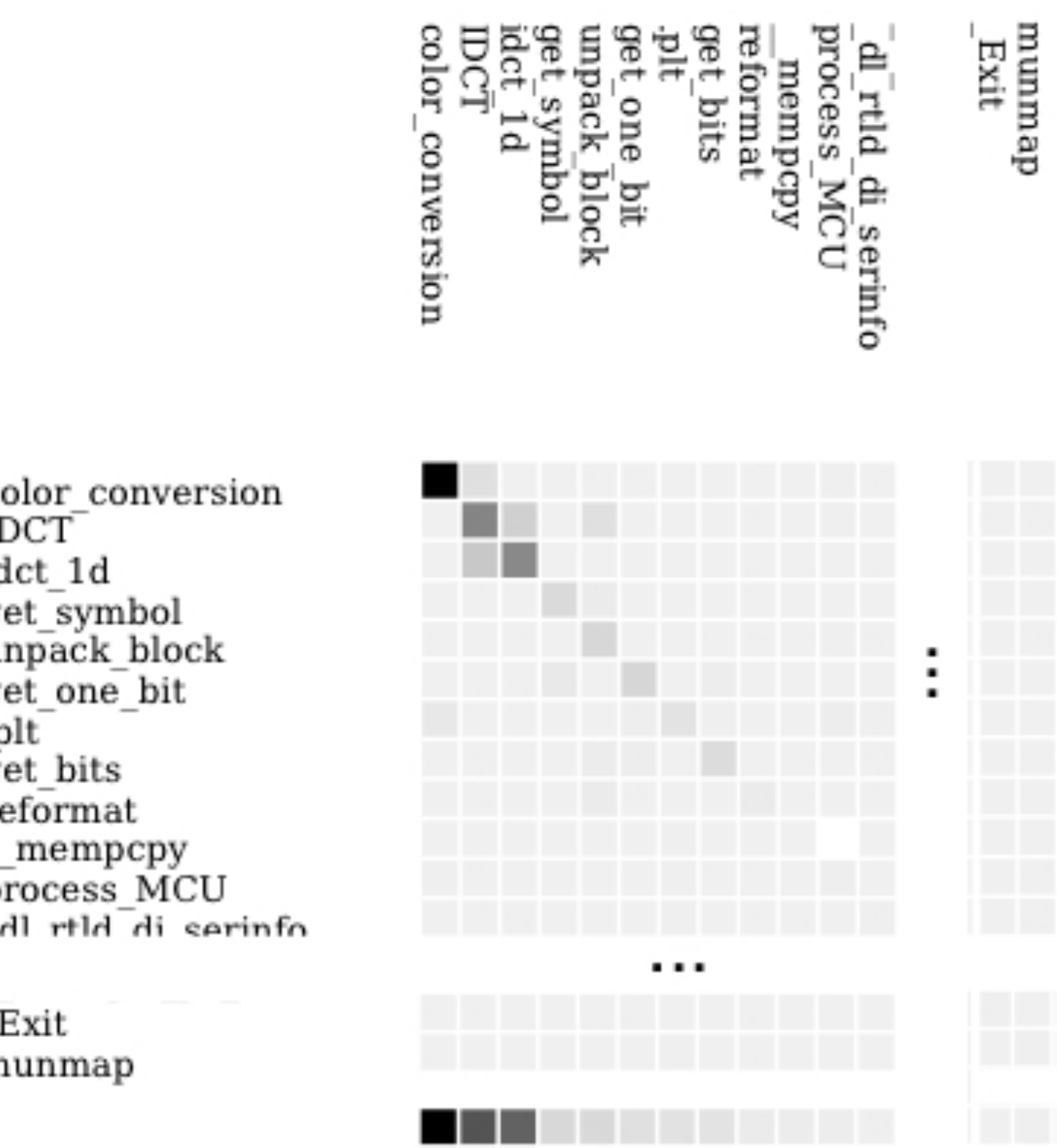
Experiment 4: Specialize Local Communication

- Specialization of local communication in addition to computation dramatically improves potential speedups

- Suggests specialized data handling and movement is a way to contend with memory wall

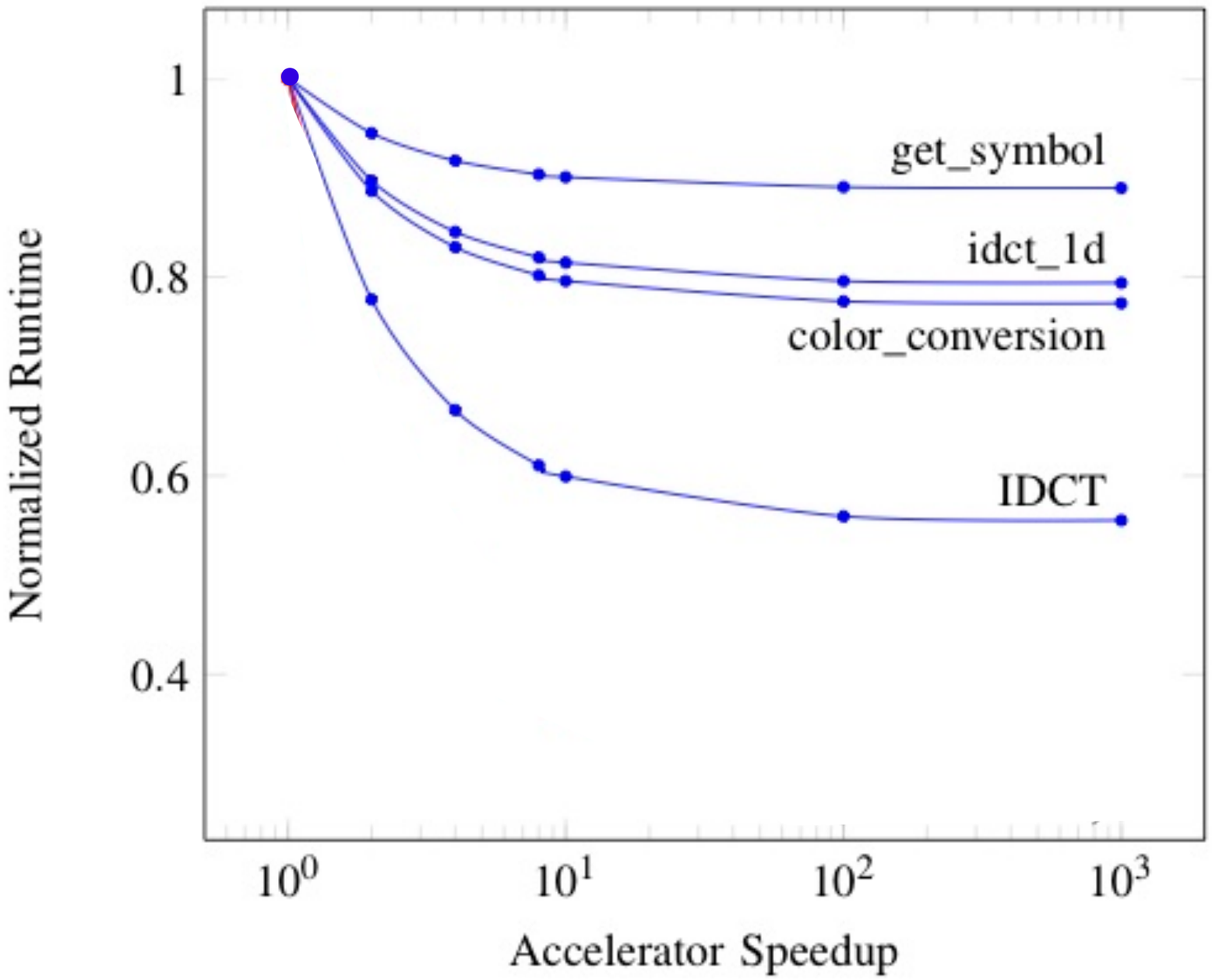


JPEG Decode Initial Results



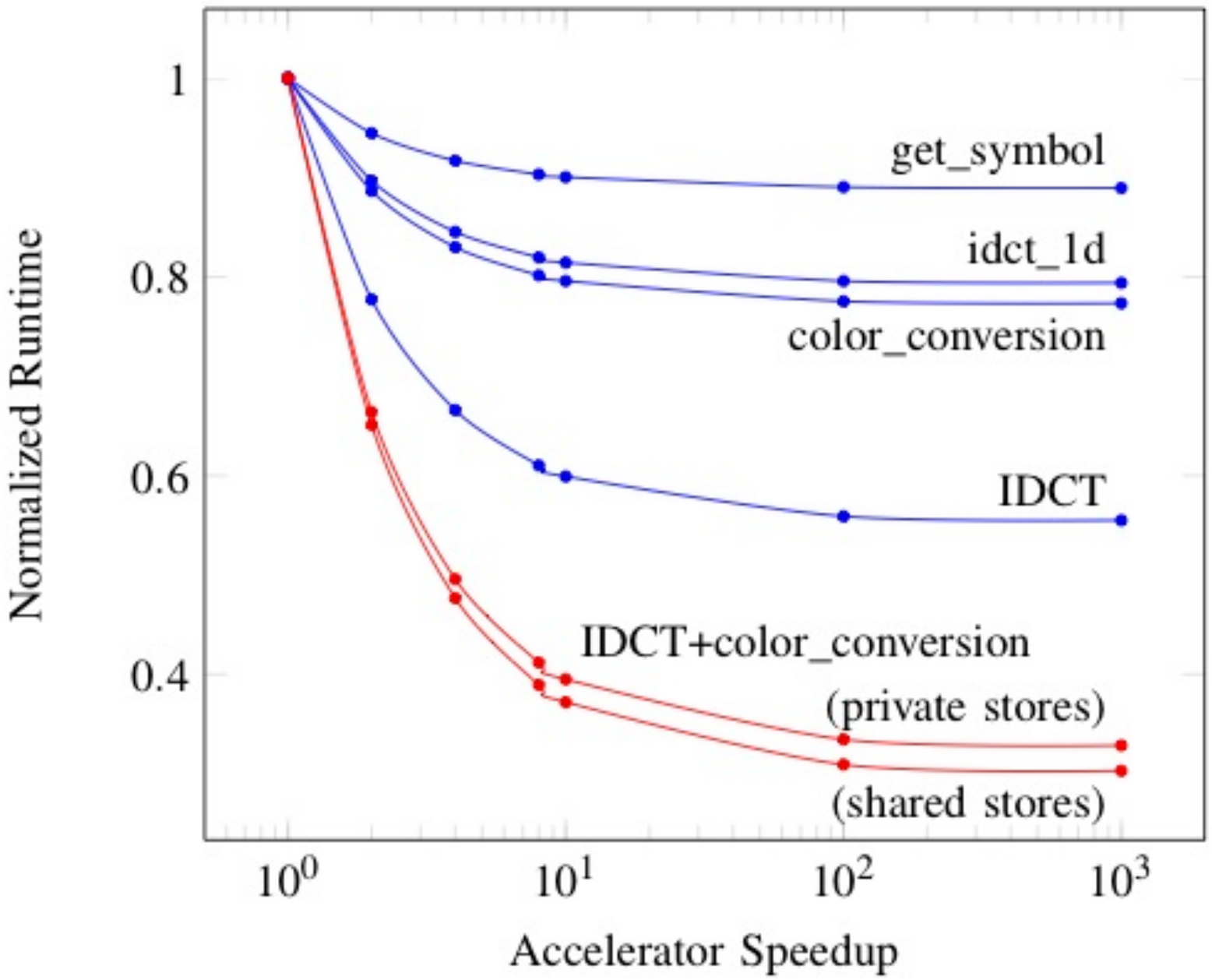
Accelerator Architectures for JPEG Decode

- Not much gain accelerating each of top four individually
- If you had to select one, function it would be #2 in terms of computation, IDCT.
- Accelerating two functions improves gains, but we see little value in a conjoined accelerator design, rather than two individuals



Accelerator Architectures for JPEG Decode

- Not much gain accelerating each of top four individually
- If you had to select one, function it would be #2 in terms of computation, IDCT.
- Accelerating two functions improves gains, but we see little value in a conjoined accelerator design, rather than two individuals



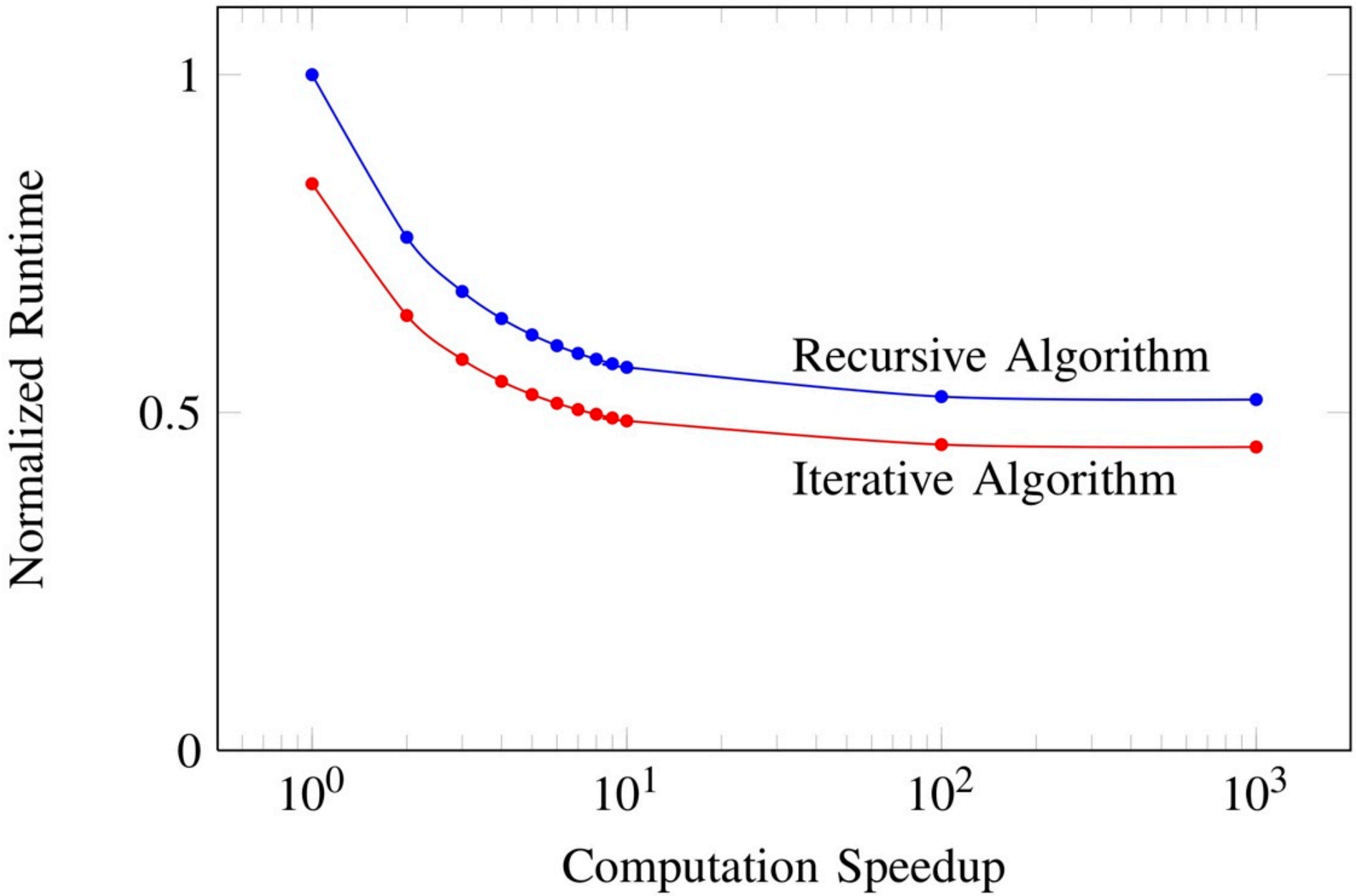
Future Work and Conclusions

- Validate model against hardware (SVM library)
- Improve efficiency of pintool (move to online version, not dependent on logs)
- Examine larger, real applications

- *Future power and performance needs are pointing in the direction of hardware specialization*
- *Understanding the interplay between an application and abstract architectural parameters can yield important design insights and performance targets*

Thank You

martha@cs.columbia.edu

BACKUP

Experiment 3: The Memory Wall