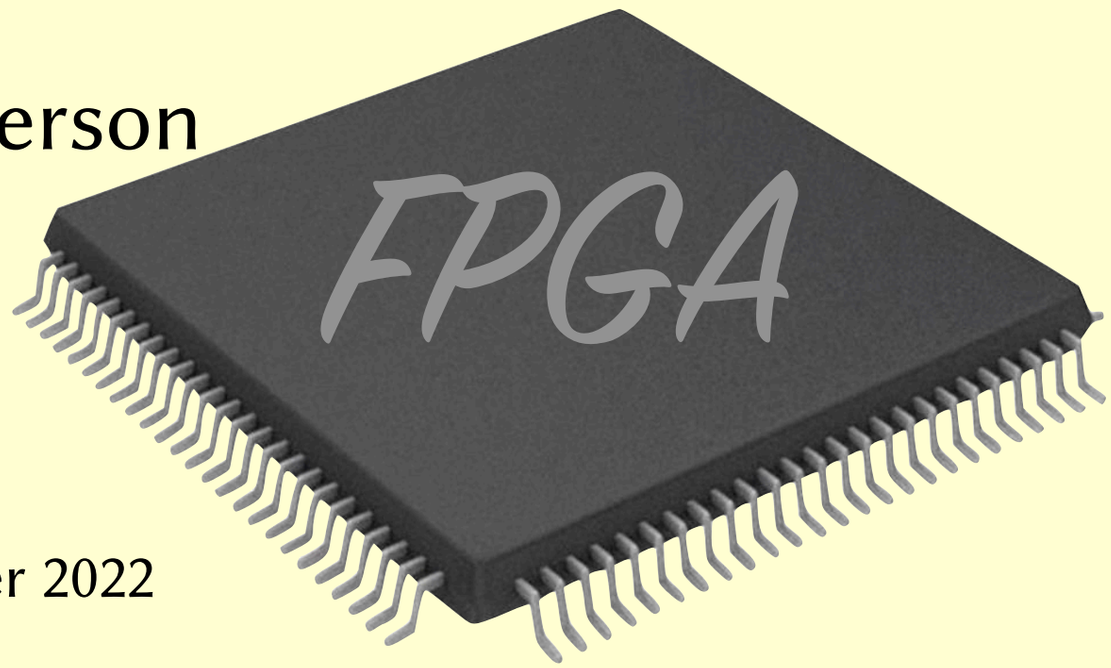
Formal methods for FPGAs

John Wickerson

5th September 2022

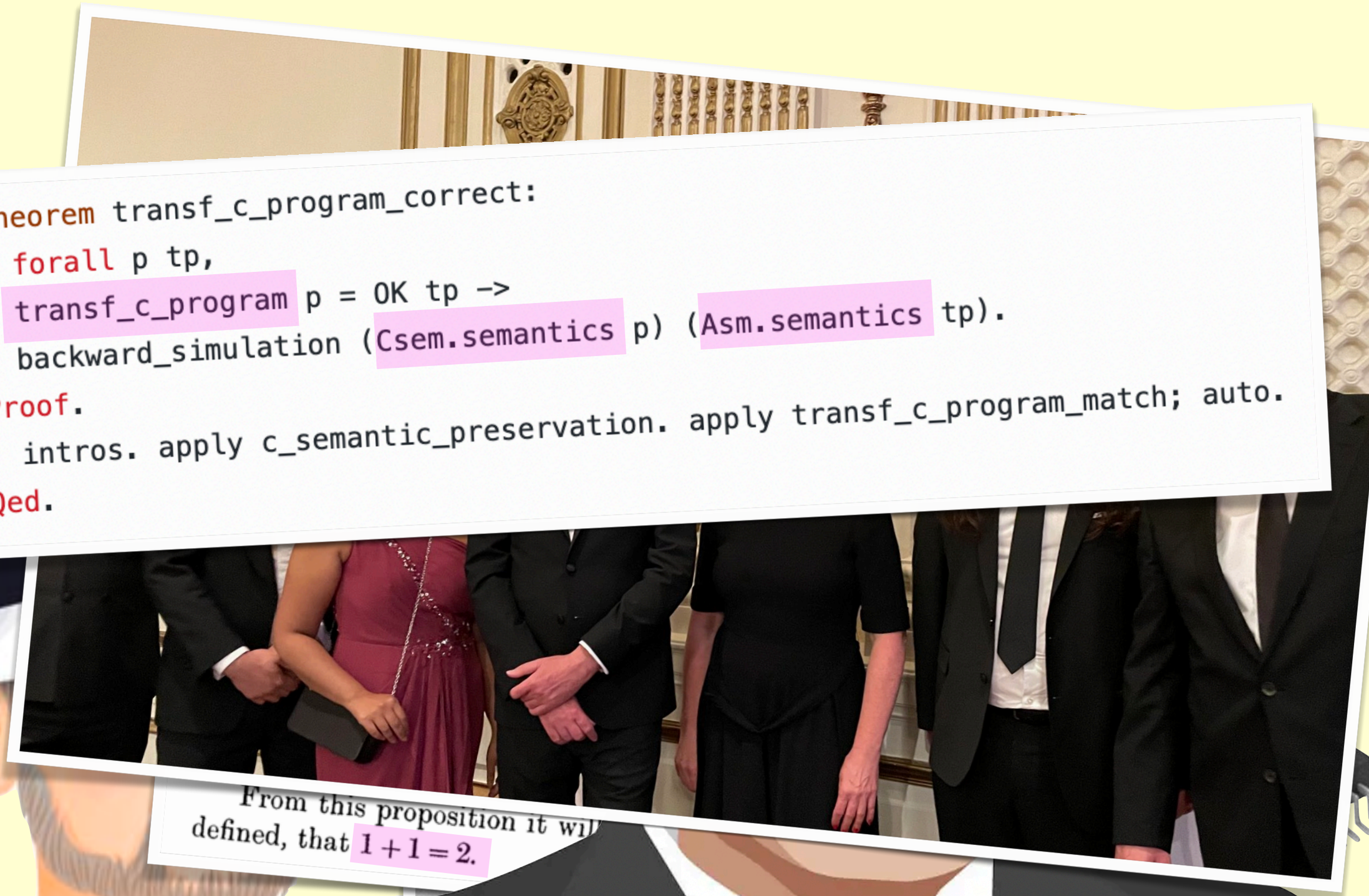
Formal methods for FPGAs

John Wickerson



September 2022

```
Theorem transf_c_program_correct:
  forall p tp,
  transf_c_program p = OK tp ->
  backward_simulation (Csem.semantics p) (Asm.semantics tp).
Proof.
  intros. apply c_semantic_preservation. apply transf_c_program_match; auto.
Qed.
```



From this proposition it wil
defined, that $1+1=2.$

"All mathem
manipulati
to t

Formal methods for...

- specifying the semantics of CPU/FPGA devices

  work led by **Dan Iorga**, with Alastair Donaldson

- proven-correct high-level synthesis

  work led by **Yann Herklotz**

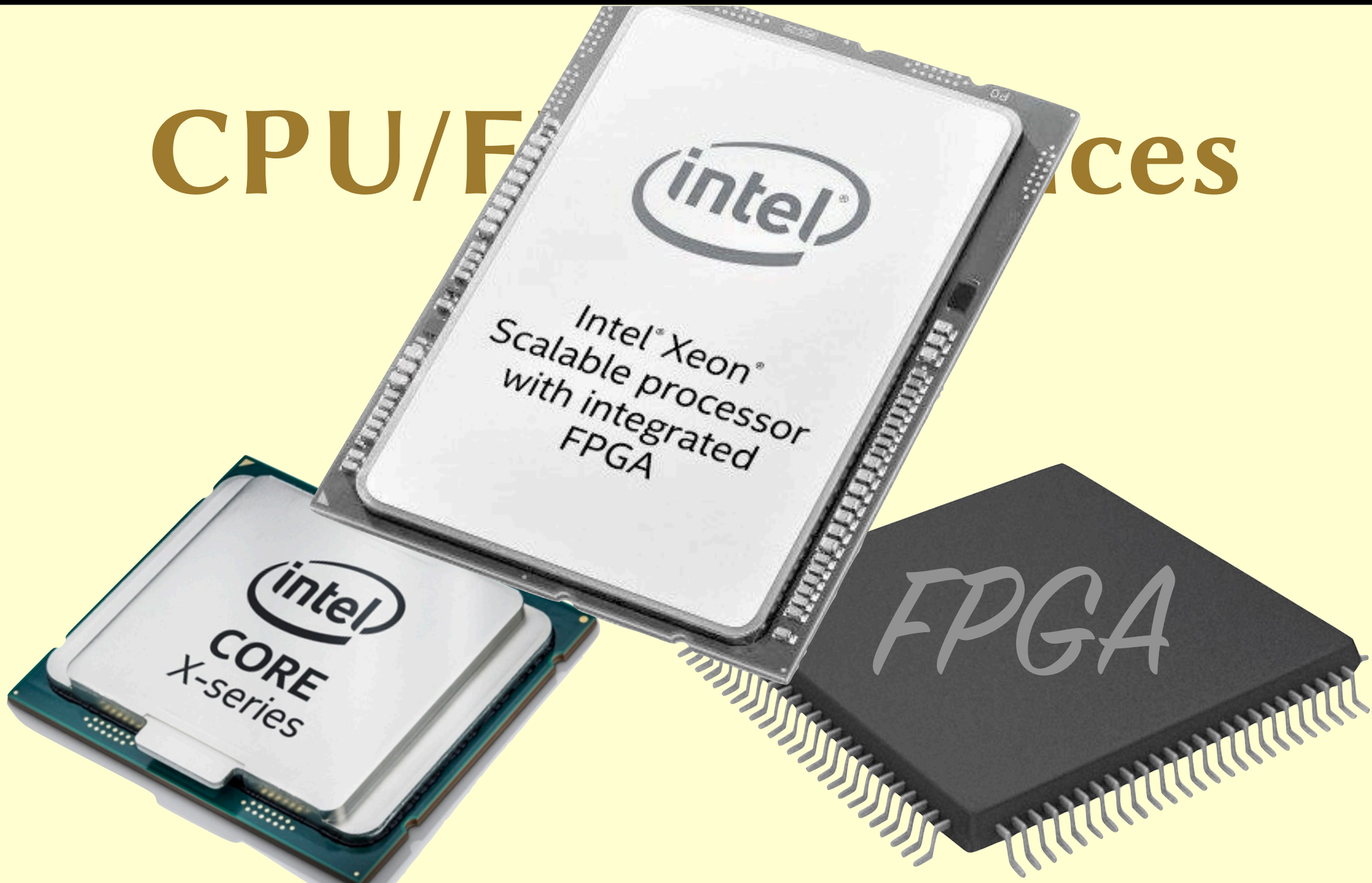
- more efficient high-level synthesis

  work led by **Jianyi Cheng**, with George Constantinides

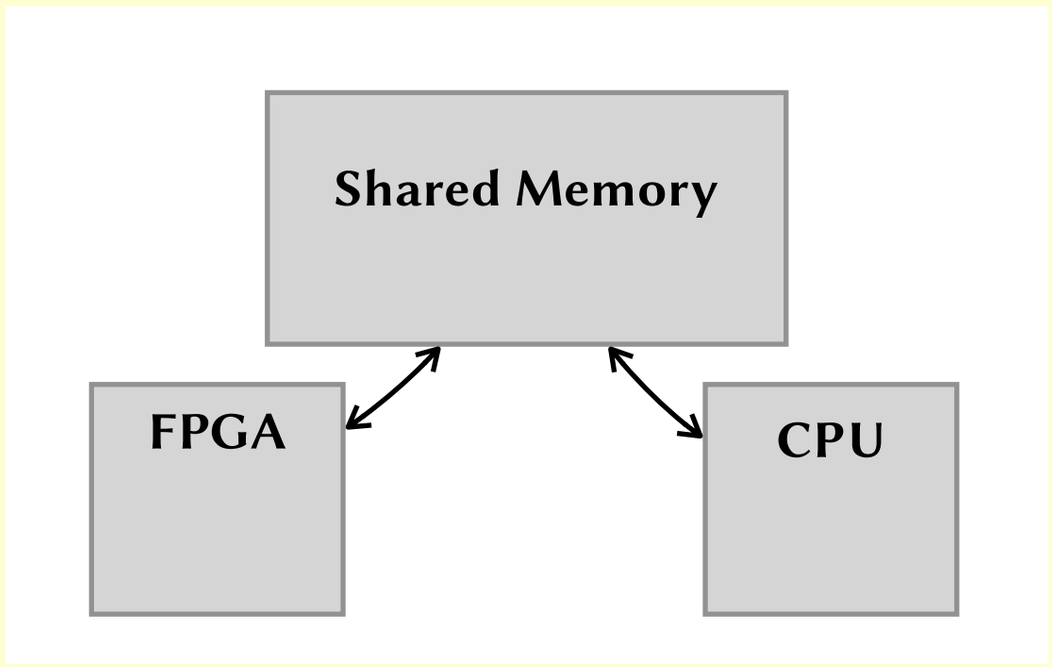
- high-level synthesis of weak-memory concurrency

  work led by **Nadesh Ramanathan**, with George Constantinides

CPU/F ces

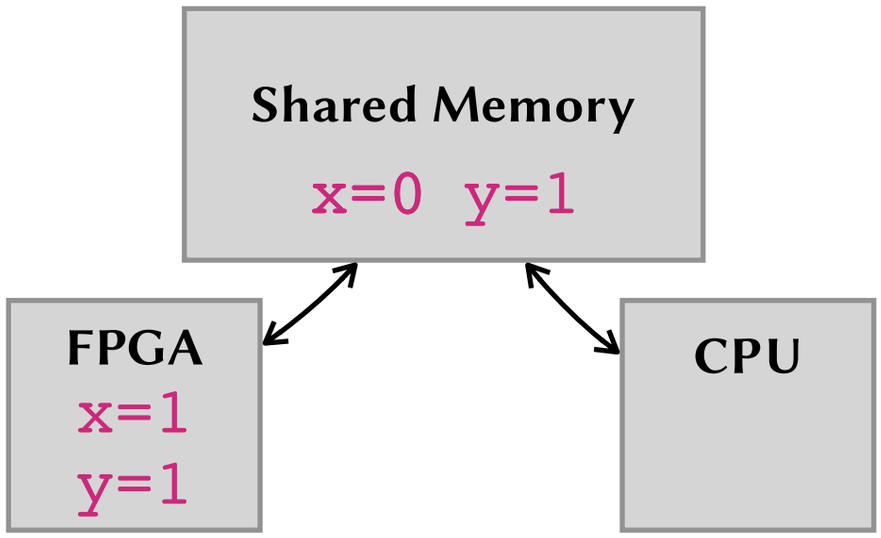

intel
Intel® Xeon®
Scalable processor
with integrated
FPGA
intel
CORE
X-series
FPGA

CPU/FPGA devices



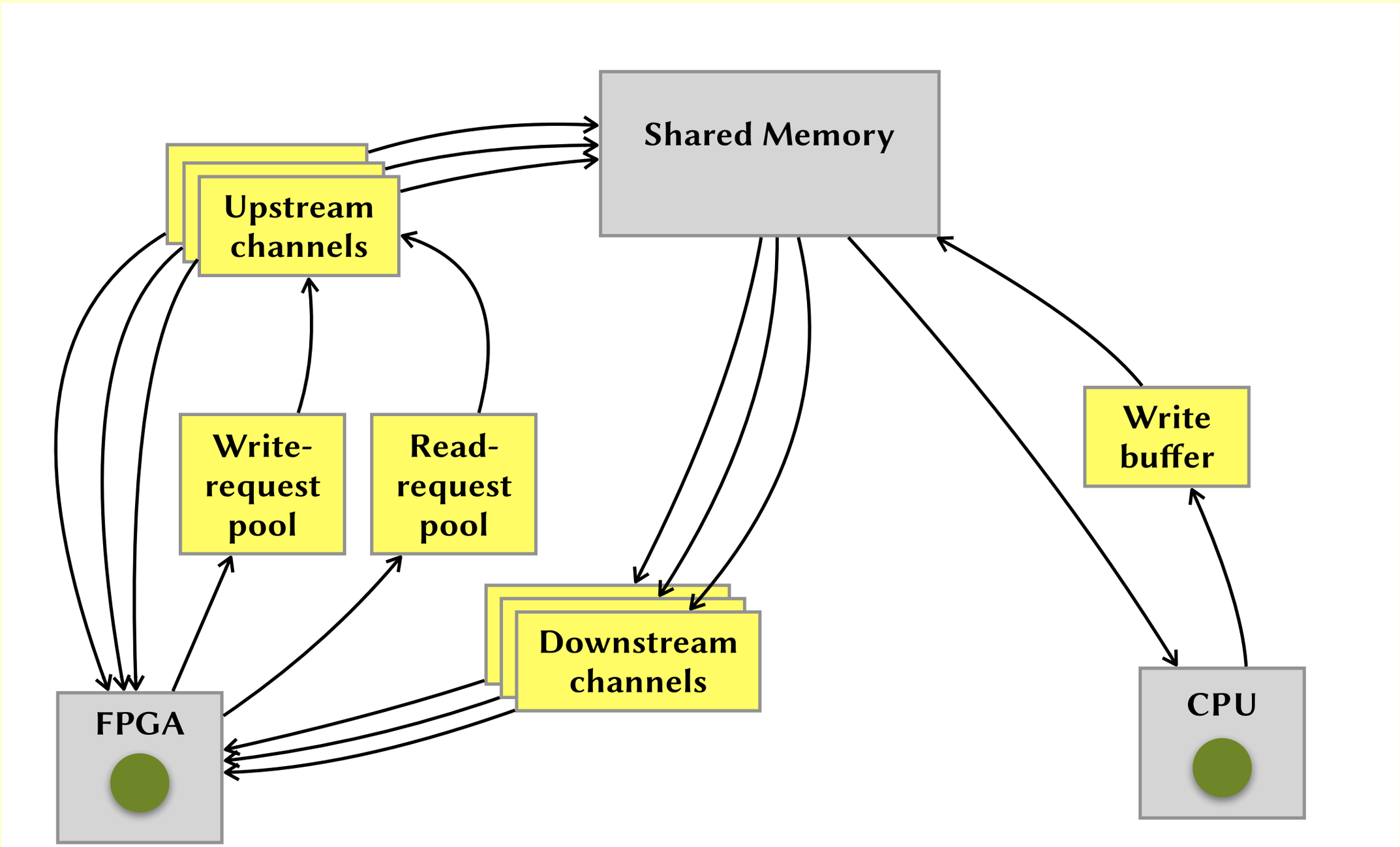

```
//FPGA:  ||  //CPU:
x=1;       ||  print(y);
y=1;       ||  print(x);
```

CPU/F ces




```
//FPGA:  ||  //CPU:
x=1;       ||  print(y);
y=1;       ||  print(x);
```

CPU/FPGA devices



Write Request

$$(WP, RP, UB, DB, SM) \xrightarrow[\text{FPGA}]{\texttt{WrReq}(c,l,v,m)} (WP +\!\!+ (\mathrm{W}, c, l, v, m), RP, UB, DB, SM)$$

Read Request

$$(WP, RP, UB, DB, SM) \xrightarrow[\text{FPGA}]{\texttt{RdReq}(c,l,m)} (WP, RP +\!\!+ (\mathrm{R}, c, l, m), UB, DB, SM)$$

Fence Request One Channel

$$(WP, RP, UB, DB, SM) \xrightarrow[\text{FPGA}]{\texttt{FnReqOne}(c,m)} (WP +\!\!+ (\mathrm{F}, c, \bot, \bot, m), RP, UB, DB, SM)$$

Fence Request All Channels

$$(WP, RP, UB, DB, SM) \xrightarrow[\text{FPGA}]{\texttt{FnReqAll}(m)} (WP +\!\!+ (\mathrm{F}, \bot, \bot, \bot, m), RP, UB, DB, SM)$$

Flush Write Request to Upstream Buffer

$$\frac{WP = head +\!\!+ (\mathrm{W}, c, l, v, m) +\!\!+ tail \qquad (\mathrm{F}, c, \_, \_, \_) \notin head \qquad (\mathrm{F}, \bot, \_, \_, \_) \notin head}{(WP, RP, UB, DB, SM) \xrightarrow[\text{FPGA}]{\texttt{WrRsp}(c,m)} (head +\!\!+ tail, RP, UB[c := UB[c] +\!\!+ (\mathrm{W}, l, v, m)], DB, SM)}$$

Write to Memory

$$\frac{UB[c] = (\mathrm{W}, l, v, m) +\!\!+ tail}{(WP, RP, UB, DB, SM) \xrightarrow[\text{FPGA}]{\tau} (WP, RP, UB[c := tail], DB, SM[l := v])}$$

Fence Response One Channel

$$\frac{WP = (\mathrm{F}, c, \bot, \bot, m) +\!\!+ tail \qquad UB[c] = \emptyset}{(WP, RP, UB, DB, SM) \xrightarrow[\text{FPGA}]{\texttt{FnRspOne}(c,m)} (tail, RP, UB, DB, SM)}$$

Fence Response All Channels

$$\frac{WP = (\mathrm{F}, \bot, \bot, \bot, m) +\!\!+ tail \qquad \forall c \in Chan.\ UB[c] = \emptyset}{(WP, RP, UB, DB, SM) \xrightarrow[\text{FPGA}]{\texttt{FnRspAll}(m)} (tail, RP, UB, DB, SM)}$$

Flush Read Request to Upstream Buffer

$$\frac{RP = head +\!\!+ (\mathrm{R}, c, l, m) +\!\!+ tail}{(WP, RP, UB, DB, SM) \xrightarrow[\text{FPGA}]{\tau} (WP, head +\!\!+ tail, UB[c := UB[c] +\!\!+ (\mathrm{R}, l, m)], DB, SM)}$$

Read from Memory

$$\frac{UB[c] = (\mathrm{R}, l, m) +\!\!+ tail \qquad SM(l) = v}{(WP, RP, UB, DB, SM) \xrightarrow[\text{FPGA}]{\tau} (WP, RP, UB[c := tail], DB[c := DB[c] +\!\!+ (l, v, m)], SM)}$$

Read Response

$$\frac{DB[c] = (l, v, m) +\!\!+ tail}{(WP, RP, UB, DB, SM) \xrightarrow[\text{FPGA}]{\texttt{RdRsp}(c,l,v,m)} (WP, RP, UB, DB[c := tail], SM)}$$

CPU Write

$$(SM, WB) \xrightarrow[\text{CPU}]{\texttt{CPUWrite}(t,l,v)} (SM, WB[t := WB[t] +\!\!+ (l, v)])$$

CPU Flush Write Buffer to Memory

$$\frac{WB[t] = (l, v) +\!\!+ tail}{(SM, WB) \xrightarrow[\text{CPU}]{\tau} (SM[l := v], WB[t := tail])}$$

CPU Fence

$$\frac{WB[t] = \emptyset}{(SM, WB) \xrightarrow[\text{CPU}]{\texttt{CPUFence}(t)} (SM, WB)}$$

CPU Read from Memory

$$\frac{SM(l) = v \qquad (l, \_) \notin WB[t]}{(SM, WB) \xrightarrow[\text{CPU}]{\texttt{CPURead}(t,l,v)} (SM, WB)}$$

CPU Read from Write Buffer

$$\frac{WB[t] = head +\!\!+ (l, v) +\!\!+ tail \qquad (l, \_) \notin tail}{(SM, WB) \xrightarrow[\text{CPU}]{\texttt{CPURead}(t,l,v)} (SM, WB)}$$

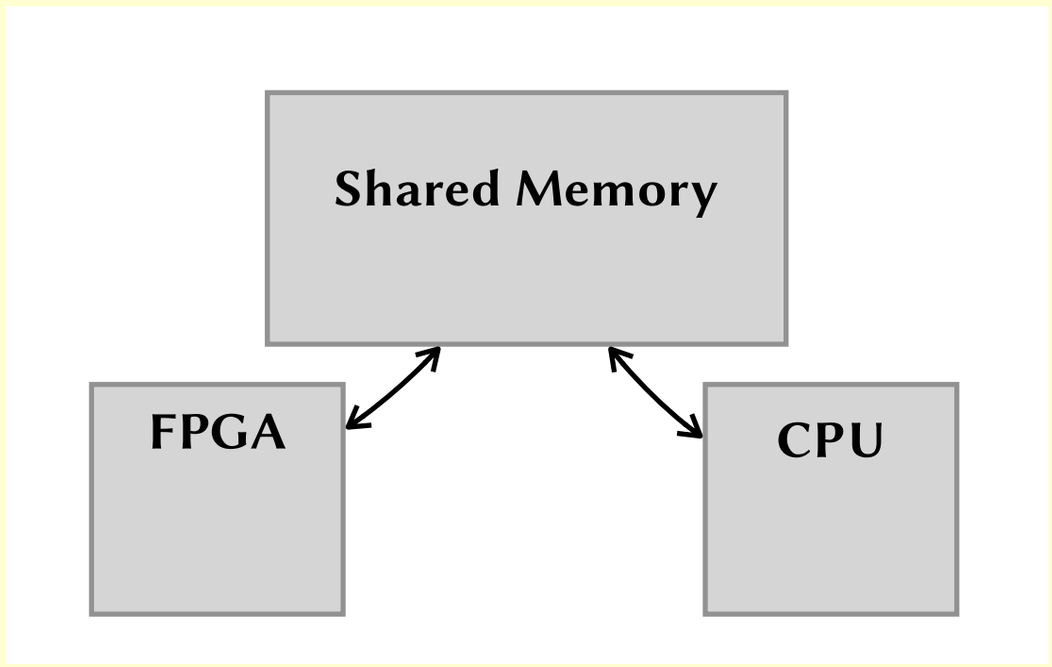
FPGA Step

$$\frac{(WP, RP, UB, DB, SM) \xrightarrow[\text{FPGA}]{a} (WP', RP', UB', DB', SM')}{(WP, RP, UB, DB, SM, WB) \xrightarrow{a} (WP', RP', UB', DB', SM', WB)}$$

CPU Step

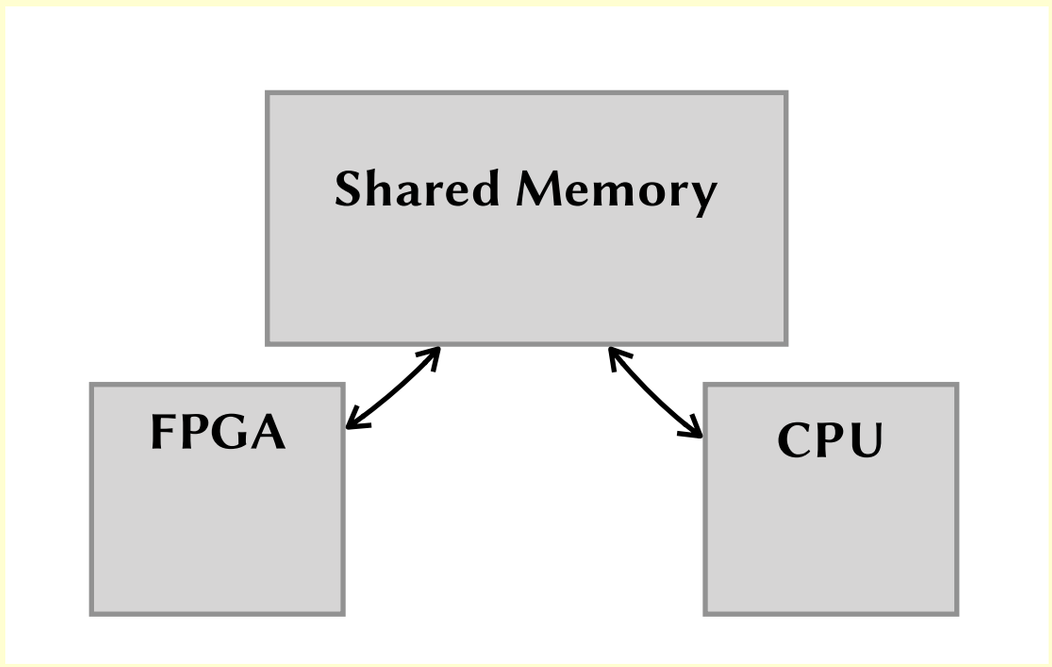
$$\frac{(SM, WB) \xrightarrow[\text{CPU}]{a} (SM', WB')}{(WP, RP, UB, DB, SM, WB) \xrightarrow{a} (WP, RP, UB, DB, SM', WB')}$$

CPU/FPGA devices




```
//FPGA:  ||  //CPU:
x=1;       ||  print(y);
y=1;       ||  print(x);
```

CPU/F ces




```
//FPGA:   ||  //CPU:
x=1;      ||  print(y);
wfence;   ||  print(x);
y=1;      ||
```

CPU/FPGA devices

- We built a model of how shared memory works in Intel CPU/FPGA devices
- Can be used as a foundation for reasoning about CPU/FPGA programs
- Can be used to automatically generate conformance tests

Formal methods for...

- specifying the semantics of CPU/FPGA devices

  work led by **Dan Iorga**, with Alastair Donaldson

- proven-correct high-level synthesis

  work led by **Yann Herklotz**

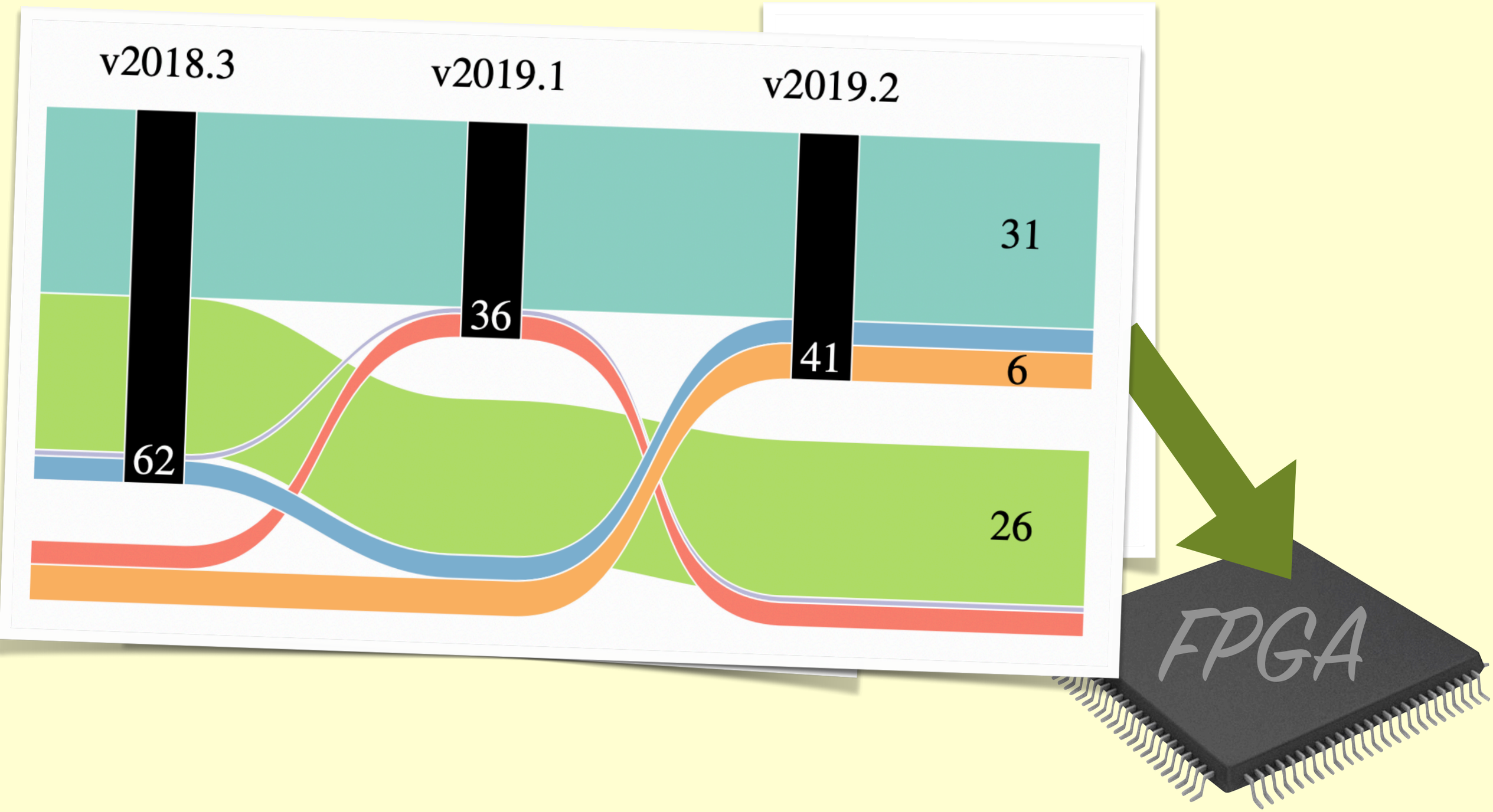
- more efficient high-level synthesis

  work led by **Jianyi Cheng**, with George Constantinides

- high-level synthesis of weak-memory concurrency

  work led by **Nadesh Ramanathan**, with George Constantinides

Proven-correct HLS



Proven-correct HLS

```
Theorem transf_c_program_correct:
  forall p tp,
  transf_c_program p = OK tp ->
  backward_simulation (Csem.semantics p) (Asm.semantics tp).
Proof.
  intros. apply c_semantic_preservation. apply transf_c_program_match; auto.
Qed.
```

C

x86

```
Theorem transf_c_program_correct:
  forall p tp,
  transf_hls p = OK tp ->
  backward_simulation (Csem.semantics p) (Verilog.semantics tp).
Proof.
  intros. apply c_semantic_preservation. apply transf_hls_match; auto.
Qed.
```

Proven-correct HLS

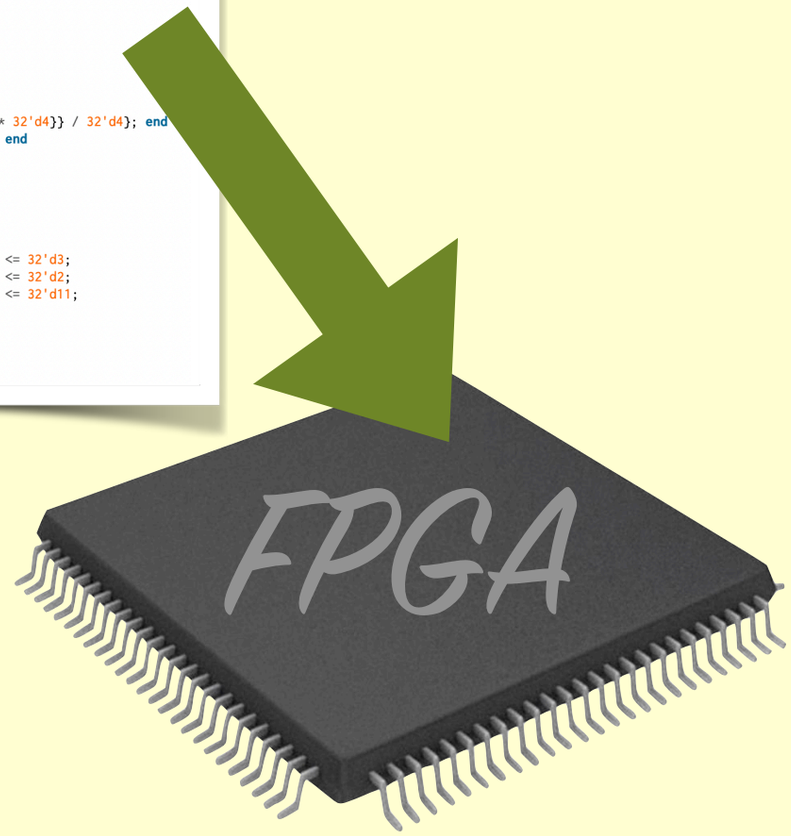
C:

```
int main() {
    int x[2] = {3, 6};
    int i = 1;
    return x[i];
}
```

HLS

Verilog:

```
module main(reset, clk, finish, return_val);
  input [0:0] reset, clk;
  output reg [0:0] finish = 0;
  output reg [31:0] return_val = 0;
  reg [31:0] reg_3 = 0, addr = 0, d_in = 0, reg_5 = 0, wr_en = 0;
  reg [0:0] en = 0, u_en = 0;
  reg [31:0] state = 0, reg_2 = 0, reg_4 = 0, d_out = 0, reg_1 = 0;
  reg [31:0] stack [1:0];
  // RAM interface
  always @(negedge clk)
    if ({u_en != en}) begin
      if (wr_en) stack[addr] <= d_in;
      else d_out <= stack[addr];
      en <= u_en;
    end
  // Data-path
  always @(posedge clk)
    case (state)
      32'd11: reg_2 <= d_out;
      32'd8: reg_5 <= 32'd3;
      32'd7: begin u_en <= ( ~ u_en); wr_en <= 32'd1;
                   d_in <= reg_5; addr <= 32'd0; end
      32'd6: reg_4 <= 32'd6;
      32'd5: begin u_en <= ( ~ u_en); wr_en <= 32'd1;
                   d_in <= reg_4; addr <= 32'd1; end
      32'd4: reg_1 <= 32'd1;
      32'd3: reg_3 <= 32'd0;
      32'd2: begin u_en <= ( ~ u_en); wr_en <= 32'd0;
                   addr <= {{{reg_3 + 32'd0} + {reg_1 * 32'd4}} / 32'd4}; end
      32'd1: begin finish = 32'd1; return_val = reg_2; end
      default: ;
    endcase
  // Control logic
  always @(posedge clk)
    if ({reset == 32'd1}) state <= 32'd8;
    else case (state)
           32'd11: state <= 32'd1;        32'd4: state <= 32'd3;
           32'd8: state <= 32'd7;         32'd3: state <= 32'd2;
           32'd7: state <= 32'd6;         32'd2: state <= 32'd11;
           32'd6: state <= 32'd5;         32'd1: ;
           32'd5: state <= 32'd4;         default: ;
         endcase
endmodule
```



sequential semantics

byte-addressed memory

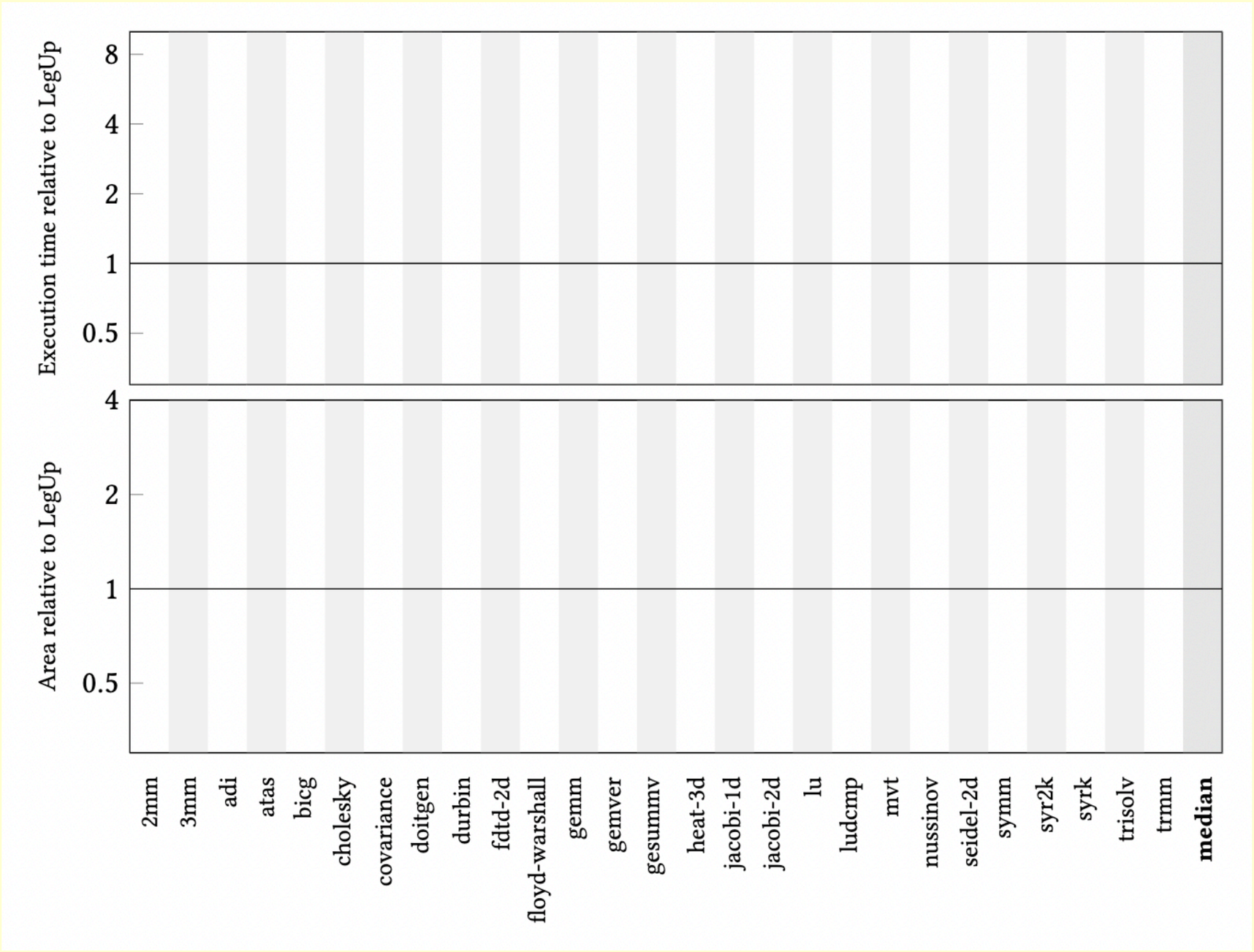
infinite memory

parallel semantics

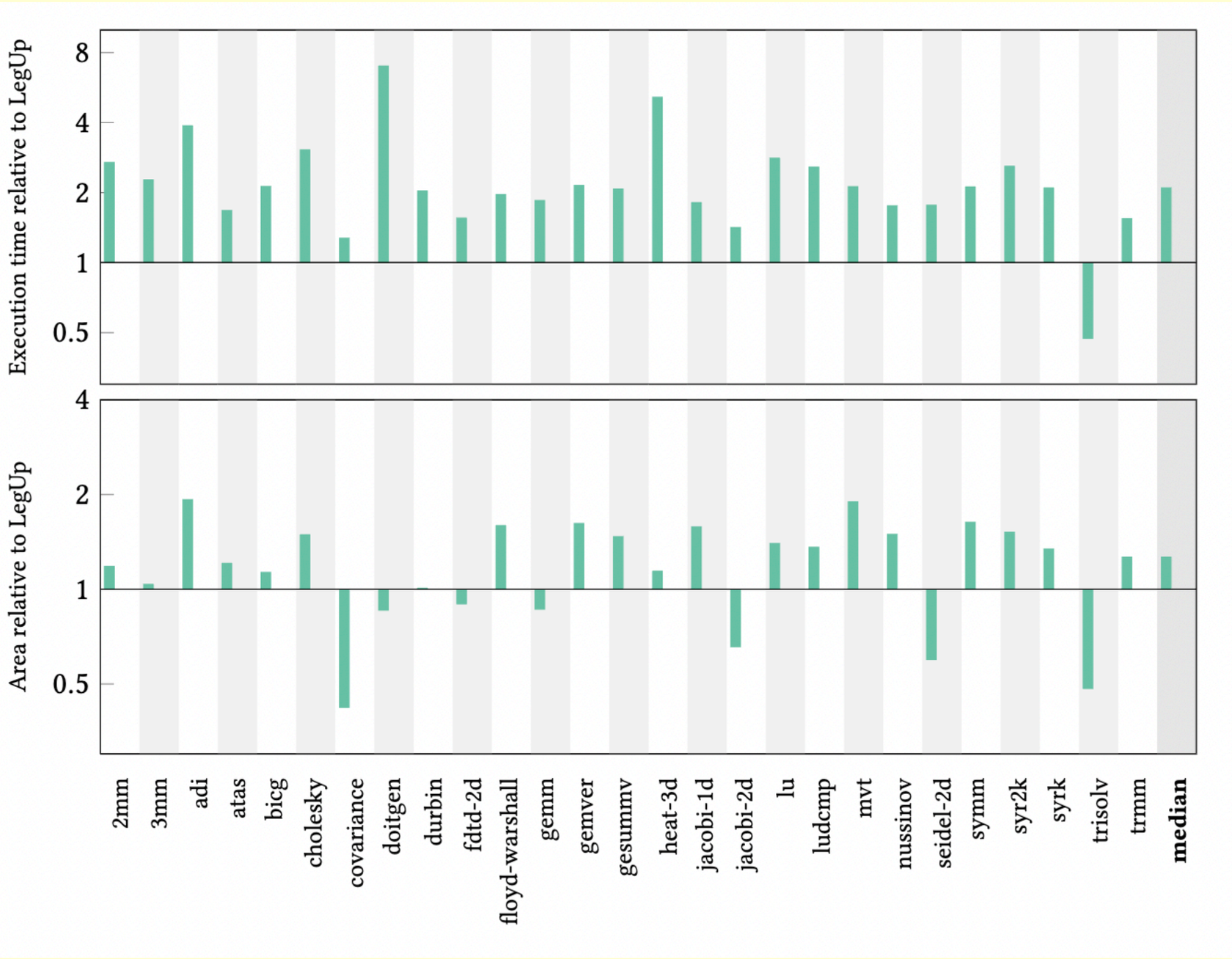
word-addressed memory

finite memory

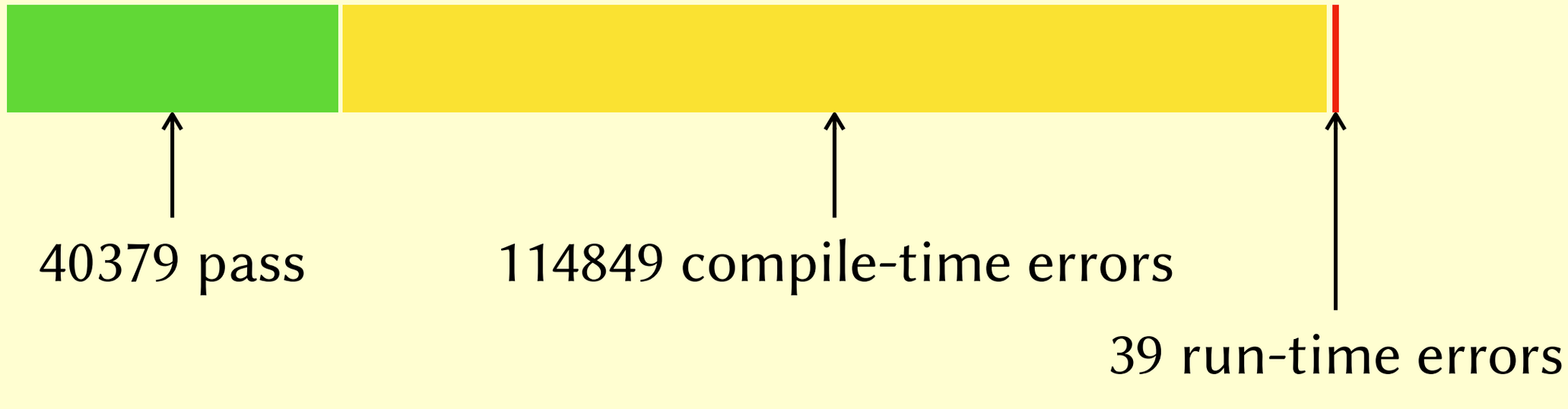
Proven-correct HLS

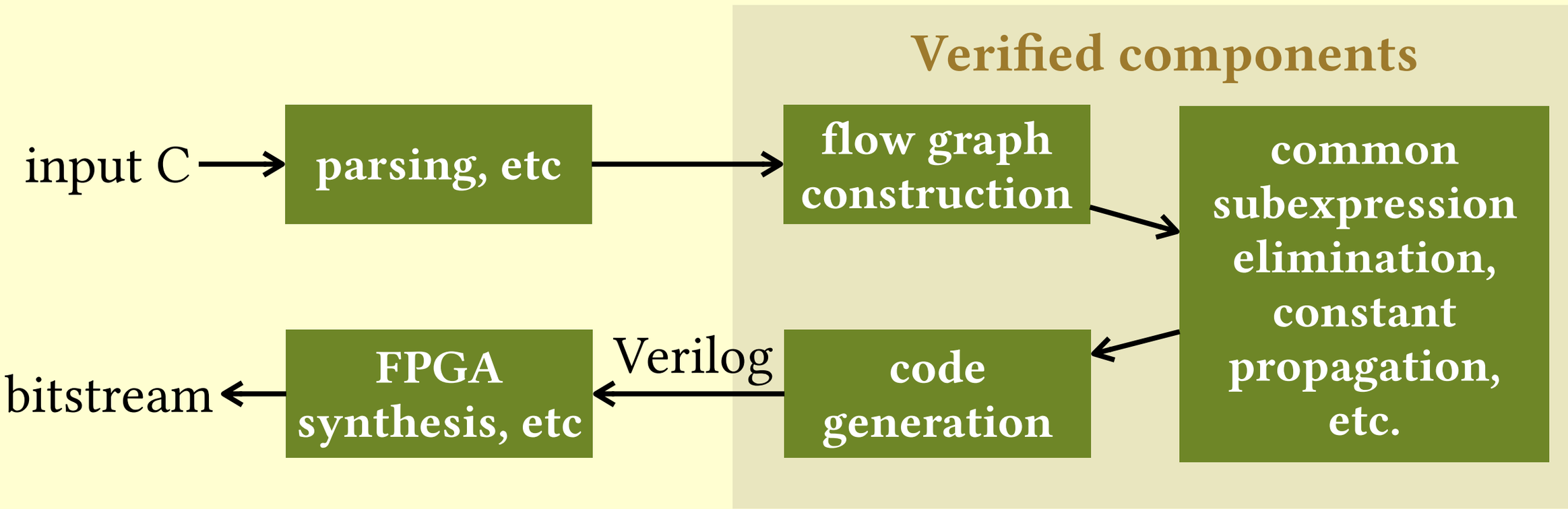


Proven-correct HLS



Proven-correct HLS





Proven-correct HLS

- We built a proven-correct HLS tool called Vericert
- Vericert is implem f assistant
- Abou st as a ied) HLS tool
- Ongoi o add more optimisations, chiefly scheduling
- Vericer pen-source and hosted on GitHub

Any questions?

Vericert