Program Synthesis for Low-Power Accelerators

Computer Science
UC Berkeley

Ras Bodik
Mangpo Phitchaya Phothilimthana
Tikhon Jelvis
Rohin Shah
Nishant Totla

What we do and talk overview

Our main expertise is in program synthesis

- a modern alternative/complement to compilation

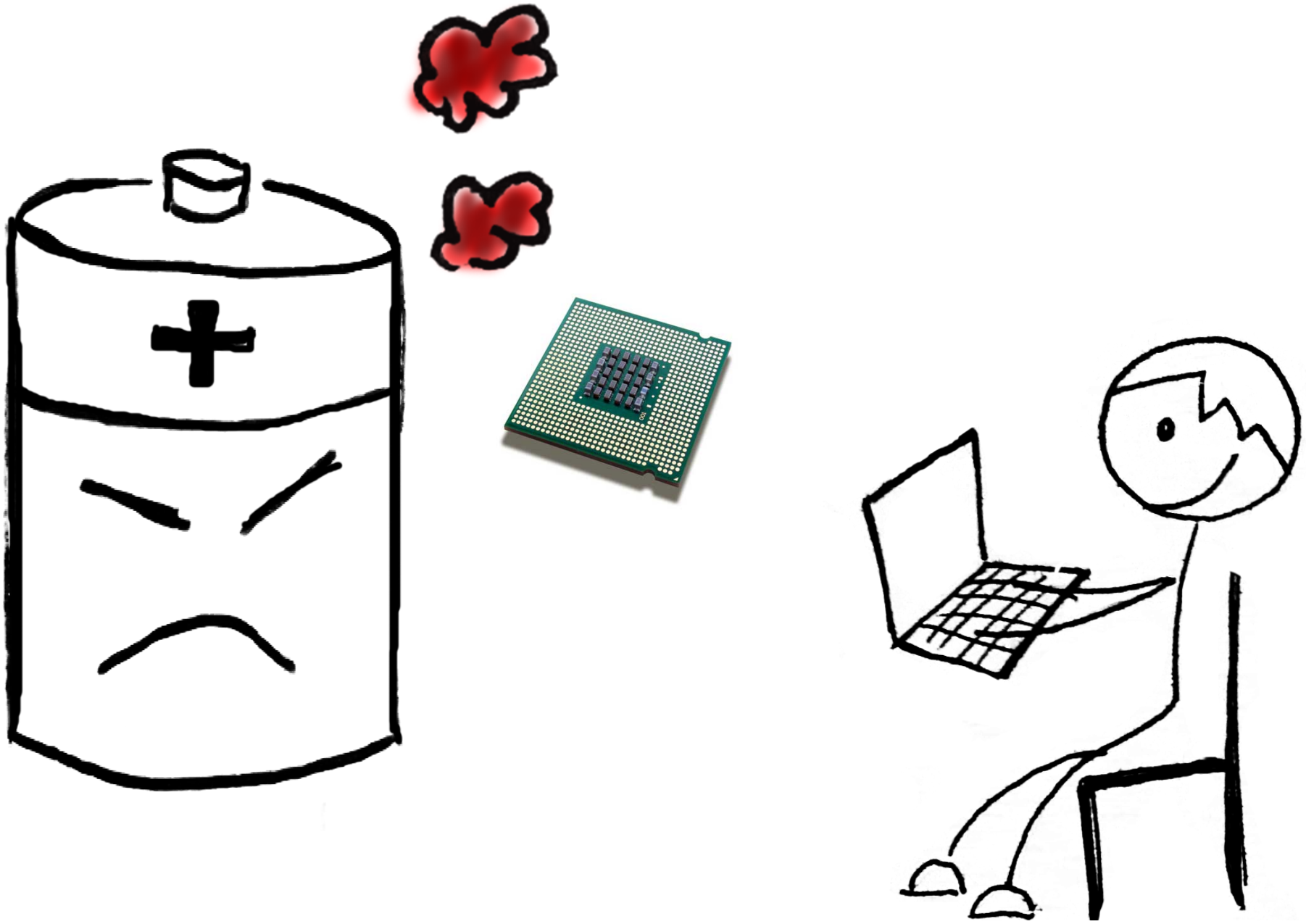
Our applications of synthesis: hard-to-write code

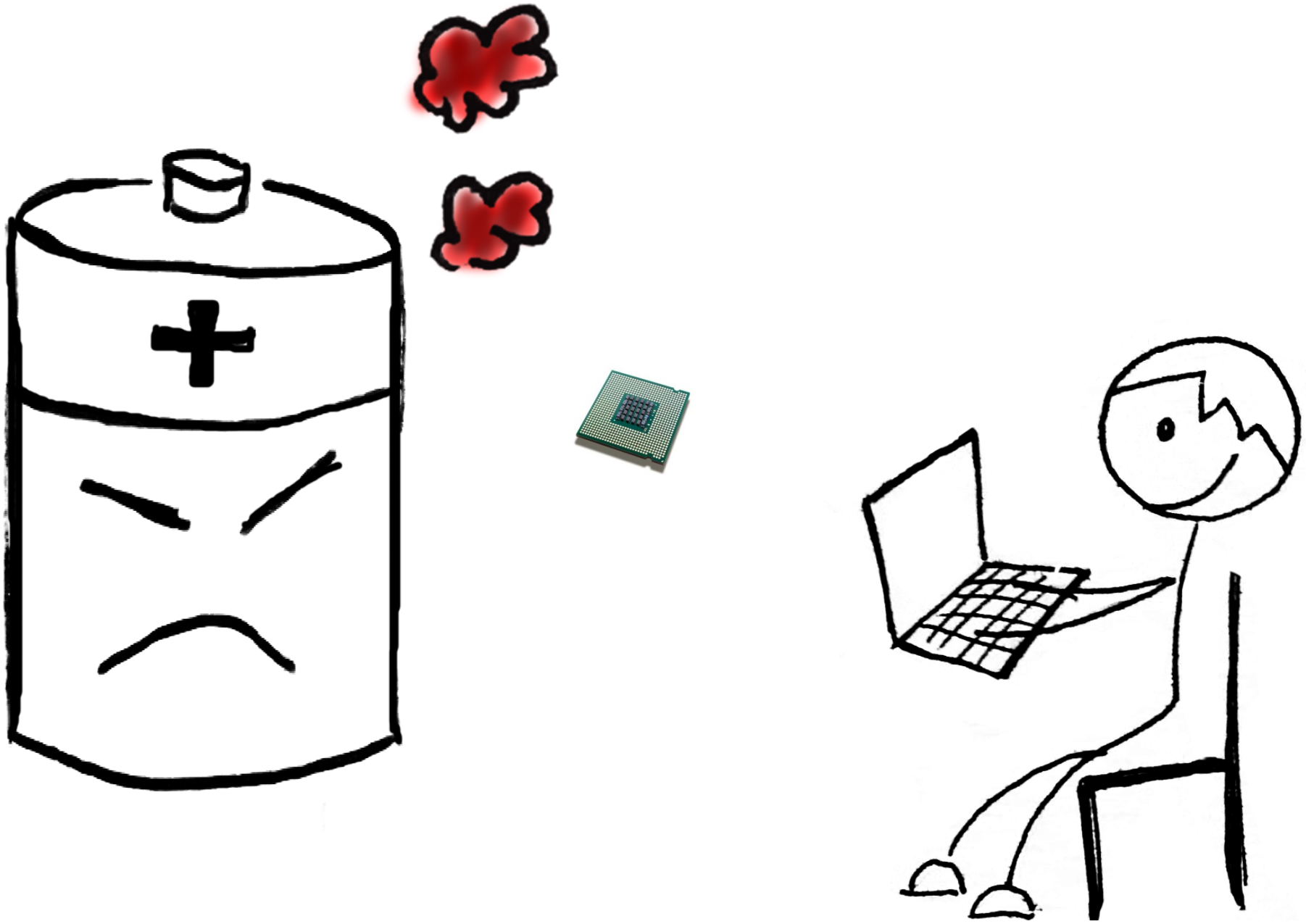
- parallel, concurrent, dynamic programming, end-user code

In this project, we explore spatial accelerators

- an accelerator programming model aided by synthesis

Future Programmable Accelerators





crazy ISA

small memory

back to 16-bit nums

no cache coherence

limited interconnect

no clock!

spatial & temporal partitioning

crazy ISA

small memory

back to 16-bit nums

no cache coherence

limited interconnect

no clock!

spatial & temporal partitioning

What we desire from programmable accelerators

We want the obvious conflicting properties:

- high performance at low energy
- easy to program, port, and autotune

Can't usually get both

- transistors that aid programmability burn energy
- ex: cache coherence, smart interconnects, …

In principle, most decisions can be done in compilers

- which would simplify the hardware
- but the compiler power has proven limited

We ask

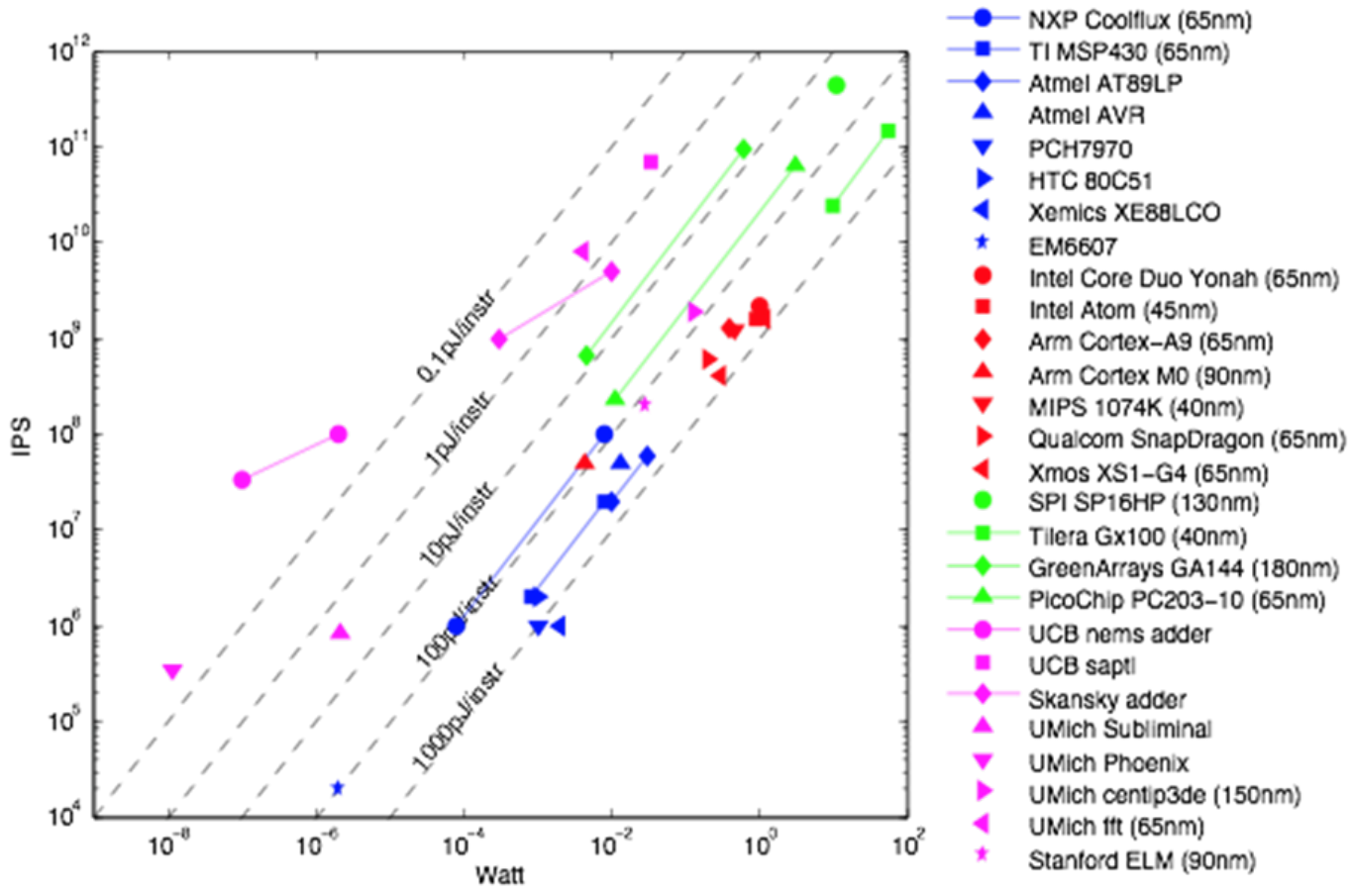
How to use fewer “programmability transistors”?

How should a programming framework support this?

Our approach: synthesis-aided programming model

GA 144: our stress-test case study

Low-Power Architectures



thanks: Per Ljung (Nokia)

Why is GA low power

The GA architecture uses several mechanisms to achieve ultra high energy efficiency, including:

- asynchronous operation (no clock tree),
- ultra compact operator encoding (4 instructions per word) to minimize fetching,
- minimal operand communication using stack architecture,
- optimized bitwidths and small local resources,
- automatic fine-grained power gating when waiting for inter-node communication, and
- node implementation minimizes communication energy.

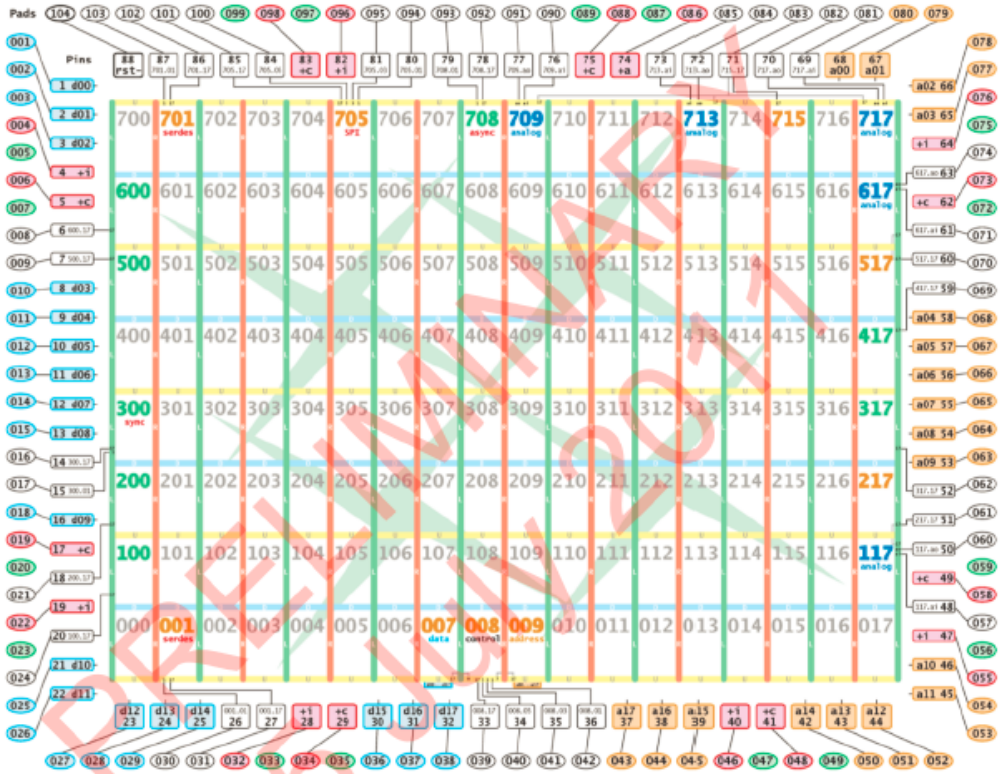
GreenArray GA144

- The GA144 is an 8 x 18 array of F18A asynchronous computers (cores)
- Cores are independent – no shared resources (i.e. clock, cache, memory bus)
  - Only globally routed signals are power/ground and reset! Very scalable architecture.
- Each core can (only!) communicate with its neighbors to the NSEW
- Cores located along perimeter include I/O functionality

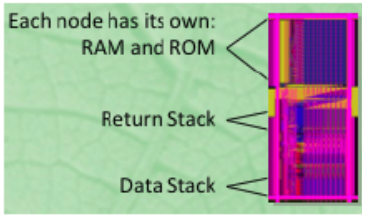
Implemented in 180nm process VDD = 1.8V

Instantaneous power usage ranges from 14 uW - 650 mW

Adjacent cores are mirror images of each other



F18A layout

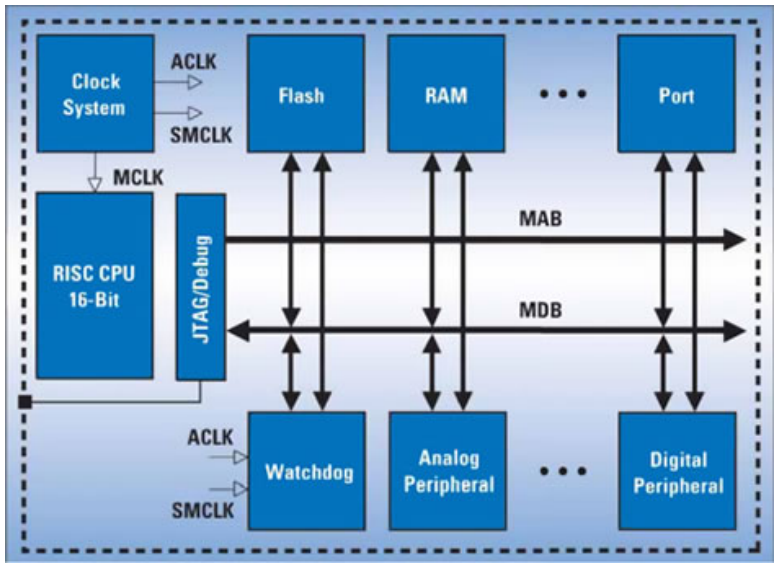


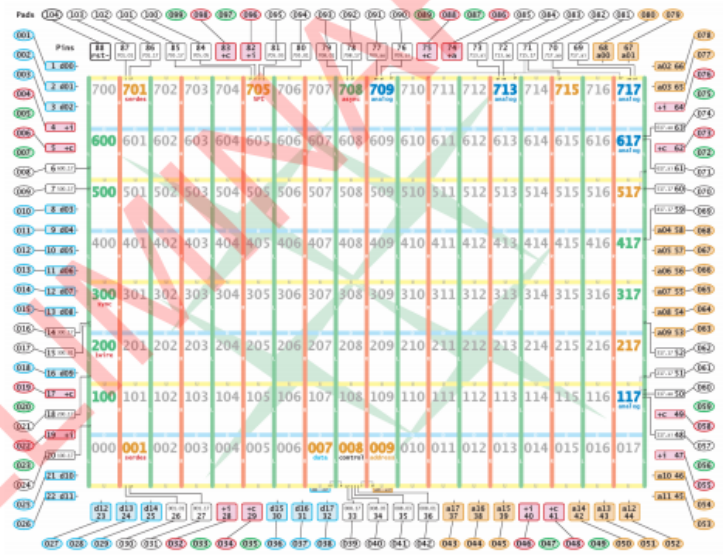
Fewer than 20k transistors per core!

Total on-chip RAM = 9216 18-bit words (equivalent amount of ROM)

slide from Rimas Avizienis

MSP430 vs GreenArray





Finite Impulse Response Benchmark

Performance	MSP430 (65nm)			GA144 (180nm)
	F2274 swmult	F2617 hwmult	F2617 hwmult+dma	
usec / FIR output	688.75	37.125	24.25	2.18
nJ / FIR output	2824.54	233.92	152.80	17.66

MSP430 **GreenArrays**

GreenArrays 144 is 11x faster and simultaneously 9x more energy efficient than MSP 430.

Data from Rimas Avizienis

How to compile to spatial architectures

Programming/compiling challenges

Partition the code and map it to cores

- also, design the algorithm for optimal partitioning

Implement inter-core communication

- don't introduce races and deadlocks

Manage distributed data structures

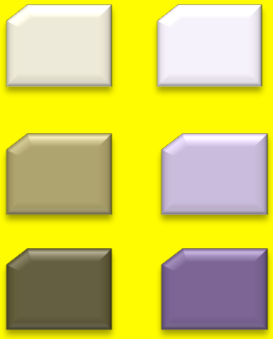
- for user-facing accelerators (GUIs), it's not just arrays

Generate efficient code

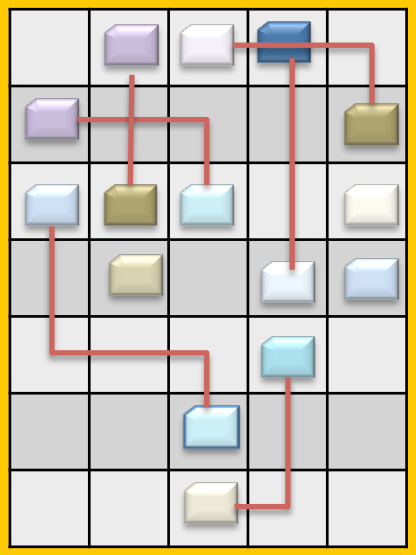
- often on a limited ISA

Programming/Compiling Challenges

Algorithm/ Pseudocode



Partition & Distribute Data Structures



Place & Route Communication

Comp1
Comp2
Comp3
Send X
Comp4
Recv Y
Comp5

Schedule & Implement code

Existing technologies

Optimizing compilers

- hard to write optimizing code transformations

FPGAs: synthesis from C

- partition C code, map it and route the communication

Bluespec: synthesis from C or SystemVerilog

- synthesizes HW, SW, and HW/SW interfaces

What does our synthesis differ?

FPGA, BlueSpec:

partition and map to minimize cost

We push synthesis further:

- super-optimization: synthesize code that meets a spec
- a different target: accelerator, not FPGA, not RTL

Benefits:

superoptimization: easy to port as there is no need to port code generator and the optimizer to a new ISA

Overview of program synthesis

Synthesis with "sketches"

Extend your language with two constructs

spec:

```
int foo (int x) {
    return x + x;
}
```

sketch:

```
int bar (int x) implements foo {
    return x << ??;
}
```

$\phi(x,y)$: $y = foo(x)$

?? substituted with an int constant meeting ϕ

result:

```
int bar (int x) implements foo {
    return x << 1;
}
```

instead of **implements**, assertions over safety properties can be used

Example: 4x4-matrix transpose with SIMD

a functional (executable) specification:

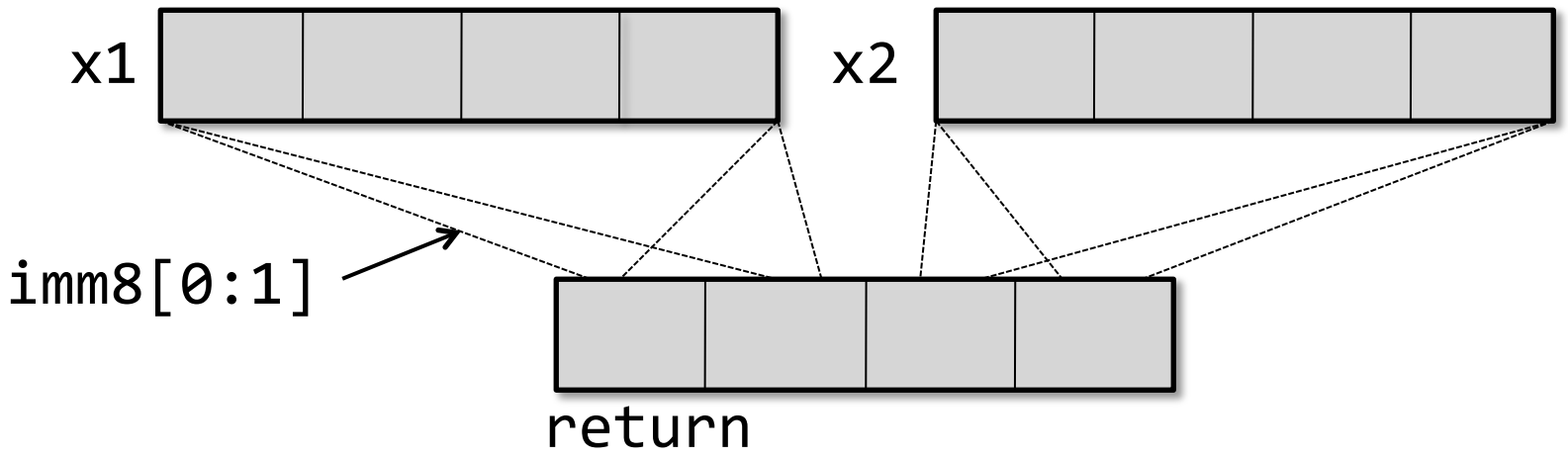
```
int[16] transpose(int[16] M) {
  int[16] T = 0;
  for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++)
      T[4 * i + j] = M[4 * j + i];
  return T;
}
```

This example comes from a Sketch grad-student contest

Implementation idea: parallelize with SIMD

Intel SHUFP (shuffle parallel scalars) SIMD instruction:

```
return = shufps(x1, x2, imm8 :: bitvector8)
```



High-level insight of the algorithm designer

Matrix M transposed in two shuffle phases

Phase 1: shuffle M into an intermediate matrix S with some number of `shufps` instructions

Phase 2: shuffle S into an result matrix T with some number of `shufps` instructions

Synthesis with partial programs helps one to complete their insight. Or prove it wrong.

The SIMD matrix transpose, sketched

```
int[16] trans_sse(int[16] M) implements trans {
  int[16] S = 0, T = 0;

  S[??::4] = shufps(M[??::4], M[??::4], ??);
  S[??::4] = shufps(M[??::4], M[??::4], ??);
  …
  S[??::4] = shufps(M[??::4], M[??::4], ??);

  T[??::4] = shufps(S[??::4], S[??::4], ??);
  T[??::4] = shufps(S[??::4], S[??::4], ??);
  …
  T[??::4] = shufps(S[??::4], S[??::4], ??);

  return T;
}
```

Phase 1

Phase 2

The SIMD matrix transpose, sketched

```
int[16] trans_sse(int[16] M) implements trans {
  int[16] S = 0, T = 0;
  repeat (??) S[??::4] = shufps(M[??::4], M[??::4], ??);
  repeat (??) T[??::4] = shufps(S[??::4], S[??::4], ??);
  return T;
}
int[16] trans_sse(int[16] M) implements trans { // synthesized code
  S[4::4]   = shufps(M[6::4],   M[2::4],  11001000b);
  S[0::4]   = shufps(M[11::4],  M[6::4],  10010110b);
  S[12::4]  = shufps(M[0::4],   M[2::4],  10001101b);
  S[8::4]   = shufps(M[8::4],   M[12::4], 11010111b);
  T[4::4]   = shufps(S[11::4],  S[1::4],  10111100b);
  T[12::4]  = shufps(S[3
  T[8::4]   = shufps(S[4
  T[0::4]   = shufps(S[1
}
```

From the contestant email:

Over the summer, I spent about 1/2 a day manually figuring it out.

Synthesis time: <5 minutes.

Demo: transpose on Sketch

Try Sketch online at http://bit.ly/sketch-language

The mechanics of program synthesis (constraint solving)

What to do with a program as a formula?

Assume a formula $S_P(x,y)$ which holds iff program P(x) outputs value y

program: `f(x) { return x + x }`

formula: $Sf(x,y)$: $y=x+x$

This formula is created as in program verification with concrete semantics [CMBC, Java Pathfinder, …]

With program as a formula, solver is versatile

Solver as an **interpreter**: given x, evaluate f(x)

$S(x,y) \wedge x=3$ *solve for* y $\boldsymbol{y \mapsto 6}$

Solver as a program **inverter**: given f(x), find x

$S(x,y) \wedge y=6$ *solve for* x $\boldsymbol{x \mapsto 3}$

This solver "bidirectionality" enables synthesis

Search of candidates as constraint solving

$SP(x,h,y)$ holds iff sketch $P[h](x)$ outputs y.

```
spec(x) { return x + x }
sketch(x) { return x << ?? }
```

$Ssketch(x,y,h)$: $y=x*2^h$

The solver computes h, thus synthesizing a program correct for the given x (here, x=2)

$Ssketch(x,y,h) \wedge x=2 \wedge y=4$ *solve for h* $\boldsymbol{h \mapsto 1}$

Sometimes h must be constrained on several inputs

$S(x1,y1,h) \wedge x1=0 \wedge y1=0 \wedge$

$S(x2,y2,h) \wedge x2=3 \wedge y2=6$ *solve for h* $\boldsymbol{h \mapsto 1}$

Inductive synthesis

Our constraints encode **inductive synthesis:**

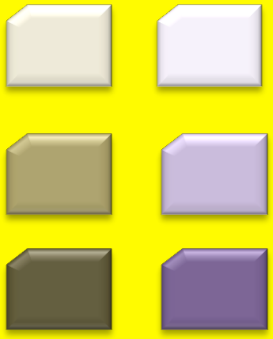
We ask for a program P correct on a few inputs.

We hope (or test, verify) that P is correct on rest of inputs.

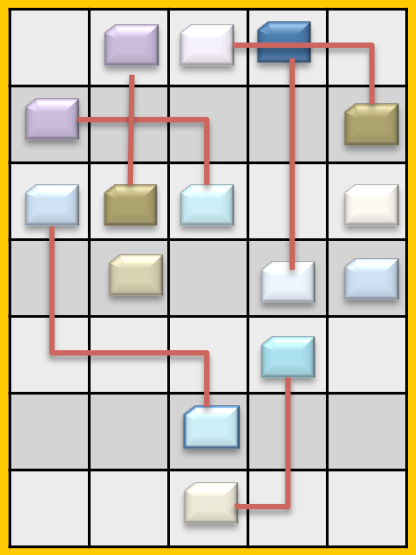
We propose a programming model for low-power devices by exploiting program synthesis.

Programming/Compiling Challenges

Algorithm/ Pseudocode



Partition & Distribute Data Structures



Place & Route Communication

Comp1
Comp2
Comp3
Send X
Comp4
Recv Y
Comp5

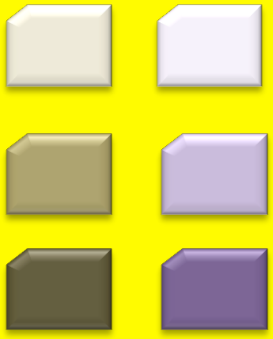
Schedule & Implement code

Project Pipeline

Language?

Language Design

- expressive
- flexible (easy to partition)



Partitioning

- minimize # of msgs
- fit each block in a core

Placement & Routing

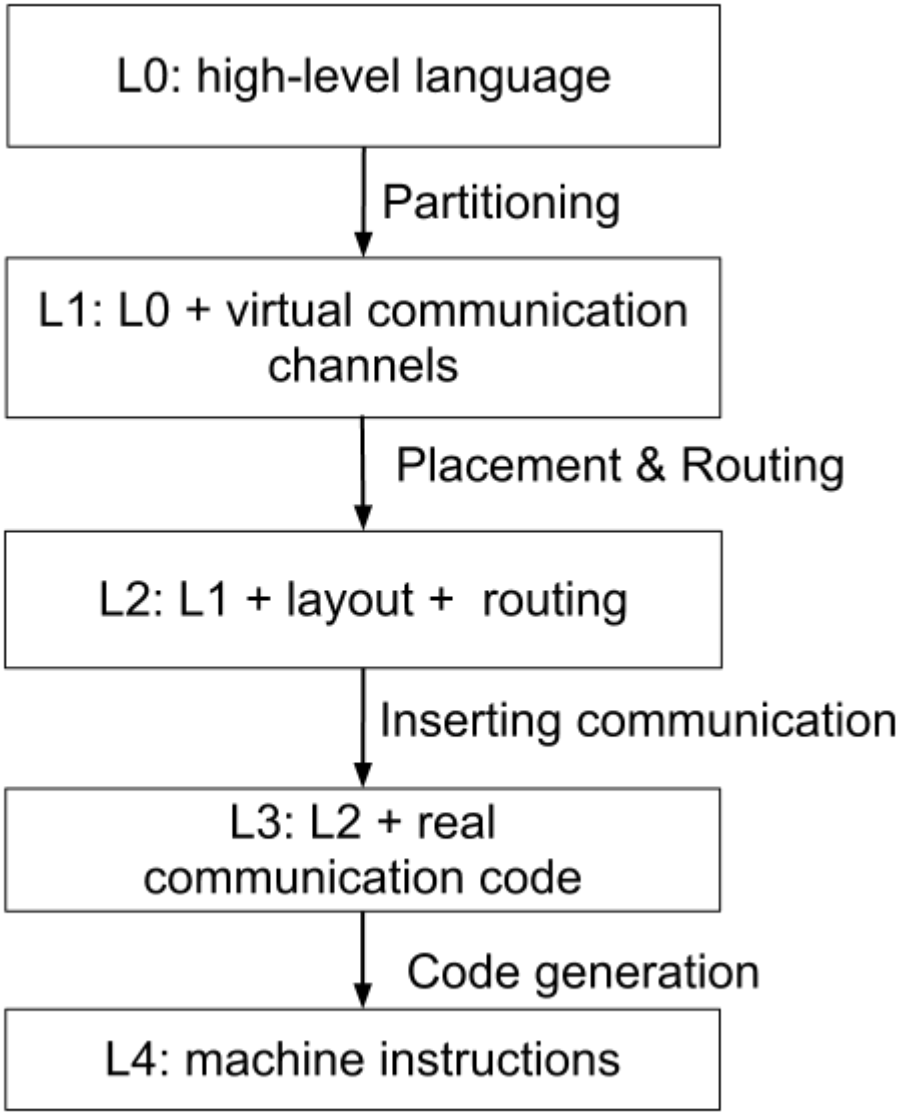
- minimize comm cost
- reason about I/O pins

Comp1
Comp2
Comp3
Send X
Comp4
Recv Y
Comp5

Intracore scheduling & Optimization

- overlap comp and comm
- avoid deadlock
- find most energy-efficient code

Programming model abstractions



MD5 case study

MD5 Hash

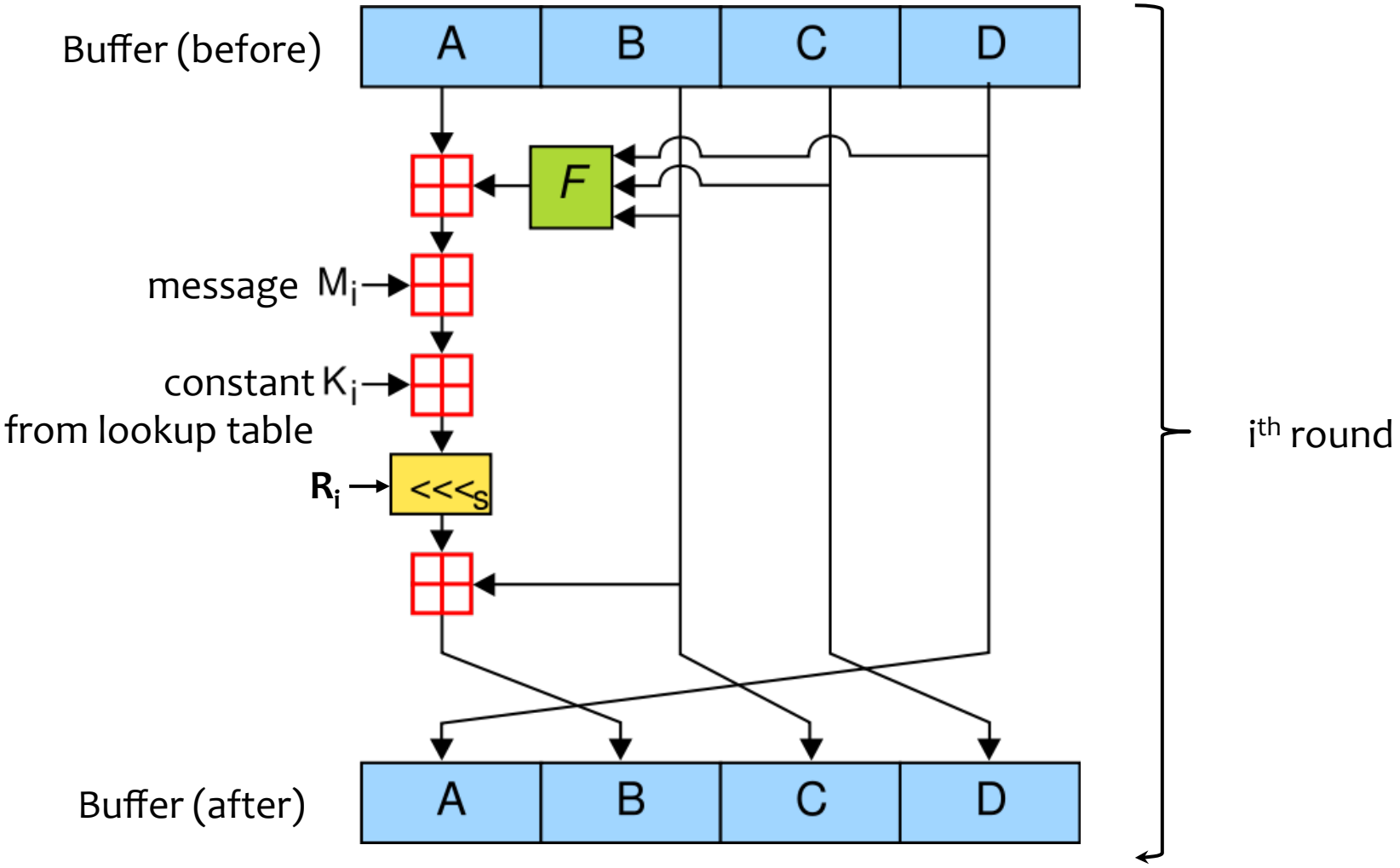


Figure taken from Wikipedia

MD5 from Wikipedia

```
//Process the message in successive 512-bit chunks:
for each 512-bit chunk of message
    break chunk into sixteen 32-bit words w[j], 0 ≤ j ≤ 15
//Initialize hash value for this chunk:
    var int a := h0
    var int b := h1
    var int c := h2
    var int d := h3
//Main loop:
    for i from 0 to 63
        if 0 ≤ i ≤ 15 then
            f := (b and c) or ((not b) and d)
            g := i
        else if 16 ≤ i ≤ 31
            f := (d and b) or ((not d) and c)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47
            f := b xor c xor d
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63
            f := c xor (b or (not d))
            g := (7×i) mod 16
        temp := d
        d := c
        c := b
        b := b + leftrotate((a + f + k[i] + w[g]) , r[i])
        a := temp
    end for
//Add this chunk's hash to result so far:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d
end for

var char digest[16] := h0 append h1 append h2 append h3 //(Output is in little-endian)

//leftrotate function definition
leftrotate (x, c)
    return (x << c) binary or (x >> (32-c));
```

Figure taken from Wikipedia

Actual MD5 Implementation on GA144

900 list
```
md5 host offset 508 0 ;
relocation helpers 32 fh load target
data
    106 +n high 4 fh load 40
      6 +n low  6 fh load 40
const and adder and rotator
    105 +n high 8 fh load 12 fh load 27
      5 +n low 10 fh load 12 fh load 36
md5
    104 +n high 14 fh load 40
      4 +n low  14 fh load 40
msg
    103 +n high 0 org 16 fh load 39
      3 +n low  0 org 16 fh load 39
rots
    102 +n high 18 fh load 3B
      2 +n low  20 fh load 3B
entry point
    101 +n high 22 fh load 1A
      1 +n low  24 fh load 40
octet feeder
      0 +n 26 fh load 40
    100 +n 28 fh load 35
    200 +n 30 fh load 3F
```

902 list
```
- load descriptor  offset 508 0 ;
inward offset 100 / 1 and drop
   if up ; then down ;
outward offset 100 / 1 and drop
   if down ; then up ;
west com @ 1 and drop if right ; then left ;
/node n offset + dup +node /ram ;

106 /node west /p   6 /node west /p
105 /node west /p   5 /node west /p
104 /node west /p   4 /node west /p
103 /node west /p   3 /node west /p
102 /node west /p   2 /node west /p
101 /node inward /p   1 /node west /p
  0 /node inward /p 100 /node outward /p
200 /node 2A /p

+break slot addr node offset + break ;
0 34 100 +break 3 3F 0 +break
```

908 list
```
- 105 high constant generator and adder
14 org
rotate 14 ni-n inward b! dup dup or
a! push push a pop pop
-if - push push @b pop !b pop then
for +* unext !b drop @b
a 2/ 2/ FFFF and or ;
+c 22 nn-n + inward b! @b . + FFFF and ;
```

910 list
```
- 5 low constant generator rotator and adder
4 org
rotate 14 ni-n inward b! dup dup or
a! push push a pop pop
-if - push !b @b pop then
for +* unext @b push !b drop pop
a 2/ 2/ FFFF and or ;
+c 22 nn-n + inward b! dup -if 2* -if
   3 !b drop FFFF and ; then
   2 !b drop FFFF and ; then
2* -if 1 !b drop FFFF and ; then
dup or !b FFFF and ;
```

912 list
```
- 105 5 constant generator rotator adder
are * 0 org
t 00 -n east b! @p .. @+ !p .. !b @b ;
 04 west a! east b!
!b .. dup or a! ..
for b @ f @ m @ . + t get +c
 push s @ rotate +c pop a! a ! next ;
* org
```

4 list
```
104 4 md5 buffer code 0 org
 0 , 0 , 0 ,
ep 04 -3 west b! 3 dup dup or a! ;
ss 08 prep for @+ !b unext ;
ab 0B prep for @b !+ unext ;
ip 0E a 3 and a! ;
   11 xyz-n push over - push and pop pop and
r 14 nn-n over FFFF or and or ;
   17 xyz-n a! push a and pop a - and +or ;
   1B xyz-n or or ;
   1C xyz-n a! push a - +or pop or ;
ss 1F west b! @b send east b! !b ;
round 23 2E a! ! dup dup or a!
15 for a @+ clip b @+ clip
dup b send c @+ clip d @+ clip
a push .. 2E f' . + pop a!
f send m pass s pass a @b 34/1 !+
@+ drop @+ drop clip next ;
md5 38
@p round f' @p round g'
@p round h' @p round ; i'
```

916 list
```
- 103 3 message buffer code 0 org
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ,
msgs 10 ni pop a! @+ @+ a push
15 for dup a! @ !b over . +
F and west a! @ !b next drop drop 1C/2 ;
msg 1C east b! @p .. @p b! @p ..
!b west !b @p .. go ..
!b @p !b @p .. !b grab .. md5 ..
west a! 3 dup push
for @ !b unext !b
msgs 1 , 0 , f's
msgs 5 , 1 , g's
msgs 3 , 5 , h's
msgs 7 , 0 , i's
@p !b .. toss ..
begin @b ! unext ;
```

918 list
```
- 102 rots generator 0 org
29E38 , B16A , 28F4B , A059 ,
send n dup east a! ! inward a! ! ;
keep 09 inward b! east a! 3 for @ !b unext ;
put 0F inward b! east a! 3 for @b ! unext ;
ncol 15 for 2/ unext
0col 17 F and ; 1col 19 3 ncol ;
2col 1B dup push 7 ncol pop
-if drop - dup then drop ;
3col 21 11 ncol - ;
jump 24 i pop + push ;
rots 25 i dup 2/ 2/ 2/ 2/ b! @b over
3 and jump 0col ; 1col ; 2col ; 3col ;
rotgen 2E east a! @p .. msg .. ! put
0 63 for dup rots send 1 . + next
drop keep ;
```

920 list
```
- 2 save and add abcd back 0 org
0 , 0 , 0 , 0 ,
0 , 0 , 0 , 0 ,
put 08 dup dup or a! 3 for
east b! @+ !b inward b! @+ !b next ;
sum 12 dup dup or a! 3 for
east b! @b @ . + dup FFFF and !+
inward b! @b @ . + over 2*
-if drop 1 . + dup then drop
FFFF and !+ next ;
sums 25 east a! @p .. msg .. 28 ! put
east a! 63 for @b ! unext sum ;
send 2F west b! dup dup or a!
7 for @+ !b unext ;
/buf 35 dup dup or a! west b!
7 for @b !+ unext 3B/3 ; 3B
```

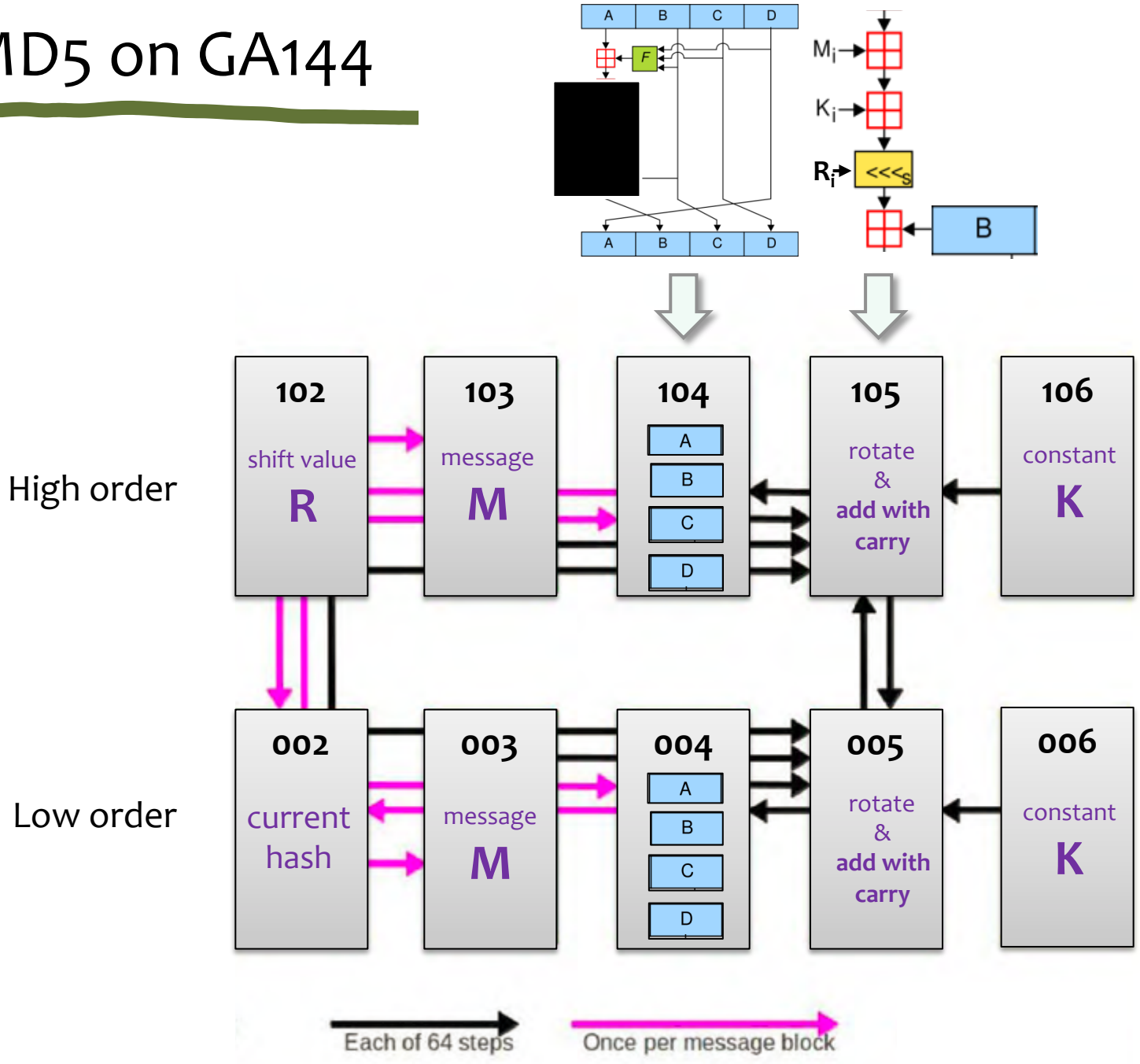
922 list
```
- 101 main control 0 org
code 102 west b! 20 for @p !b unext ..
103 dup dup or a! 15 for
@p !+ unext .. 09 gowest ; goeast ; 0B
highs 0B east b! dup dup or a!
8 for @+ !b unext a push inward a!
15 for @ !b unext pop a!
@+ !b @+ !b .. @p !b ; .. rotgen ..
```

924 list
```
- 1 main control 0 org
code 2 west b! 20 for @p !b unext ..
3 dup dup or a! 15 for
@p !+ unext .. 09 gowest ; goeast ; 0B
half 0B p push west a! @ @
7 for 2* unext or pop a! ! ;
lows 12 inward b! @p .. highs .. !b
east b! dup dup or a!
8 for @+ !b unext a push
15 for east half inward half next
pop a! @+ !b @+ !b .. @p !b ; .. sums ..
pass east b! west .. a! @p !b .. send ..
7 for @b ! unext ;
prime 2F east b! 38 a! @p .. /buf ..
!b 7 for @+ !b unext 37/3 gowest ; 38 38 org
2301 , 6745 , AB89 , EFCD ,
DCFE , 98BA , 5476 , 1032 , 40
```

926 list
```
- 0 octet feeder 0 org 0 , 0 , 0 ,
add 03 n-n @ . + dup FFFF and !+ ;
count 06 1 a! 8 add 2* -if
drop 1 add then drop ;
get 0F @ -if dup or a! 80 ; then
a if count then a! ;
octets 16 n for get !b next ;
ablk 1A n 55 octets
a if drop 7 octets ; then drop
@+ drop a for
@+ dup FF and !b 7 for 2/ unext
FF and !b next
0 dup dup or a! 3 for @ !b unext warm ;
start 30 east b! inward a!
begin @p !b .. lows .. ablk end
digest east b! @p .. pass .. !b inward a!
7 for @b ! unext 40/3 ;
```

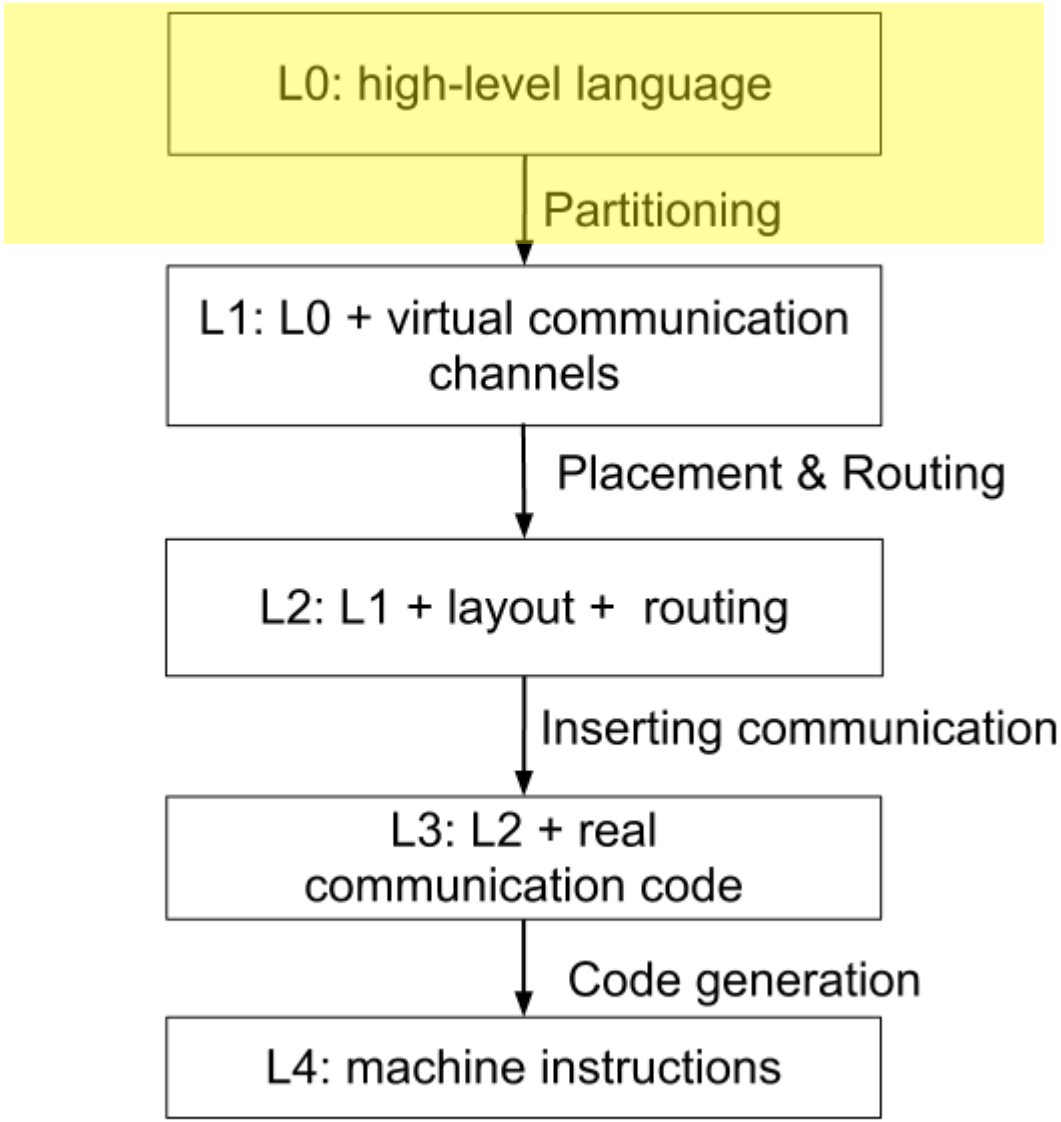
928 list
```
- 100 application interface
addresses 0-7 reserved for result 8 org
/count 08 inward b! 0 1 ..
@p !b !b dup .. @p a! @p dup ..
!b @p !b .. !+ !+ ..
/digest 10 8 a! 3 for @p !b unext ..
   east b! .. @p !b .. prime ..
/msg @p !b .. start ..
up a! @ begin !b @ -until
!b 0 a! @p .. digest .. !b
7 for @b !+ unext 24/3 warm ; .. /count ; 25
```

MD5 on GA144


A
B
C
D
F
Mi
Ki
Ri
<<<s
B
High order
Low order
102
shift value
R
103
message
M
104
A
B
C
D
105
rotate
&
add with
carry
106
constant
K
002
current
hash
003
message
M
004
A
B
C
D
005
rotate
&
add with
carry
006
constant
K
Each of 64 steps
Once per message block

This is how we express MD5

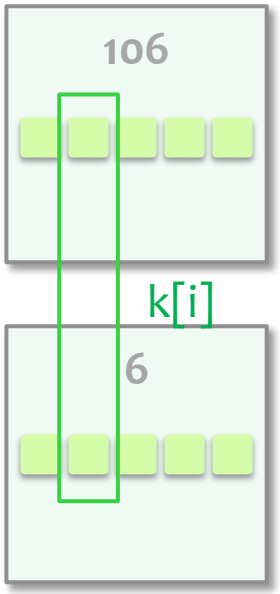
Project Pipeline



Annotation at Variable Declaration

```
typedef pair<int,int> myInt;

vector<myInt>                    message[16];
vector<myInt>                    k[64];
vector<myInt>                    output[4];
vector<myInt>                    hash[4];
vector<int>                      r[64];
```



high order

low order

`@core` indicates where data lives.

Annotation in Program

`(104,4)` is home for `md5()` function

```
void@(104,4) md5() {
  for(myInt@any t = 0; t < 16; t++) {
    myInt@here a = hash[0], b = hash[1], c = hash[2], d = hash[3];
    for(myInt@any i = 0; i < 64; i++) {
      myInt@here temp = d;
      d = c;
      c = b;
      b = round(a, b, c, d, i);
      a = temp;
    }
    hash[0] += a;
    hash[1] += b;
    hash[2] += c;
    hash[3] += d;
  }
  output[0] = hash[0];
  output[1] = hash[1];
  output[2] = hash[2];
  output[3] = hash[3];
}
```

`@any` suggests that any core can have this variable.

`@here` refers to the function's home `@(104,4)`

MD5 in CPart

```
typedef pair<int,int> myInt;

vector<myInt>@{[0:64]=(106,6)} k[64];

myInt@(105,5) sumrotate(myInt@(104,4) buffer, ...) {
  myInt@h sum = buffer +@here k[i] + message[g];
  ...
}
```

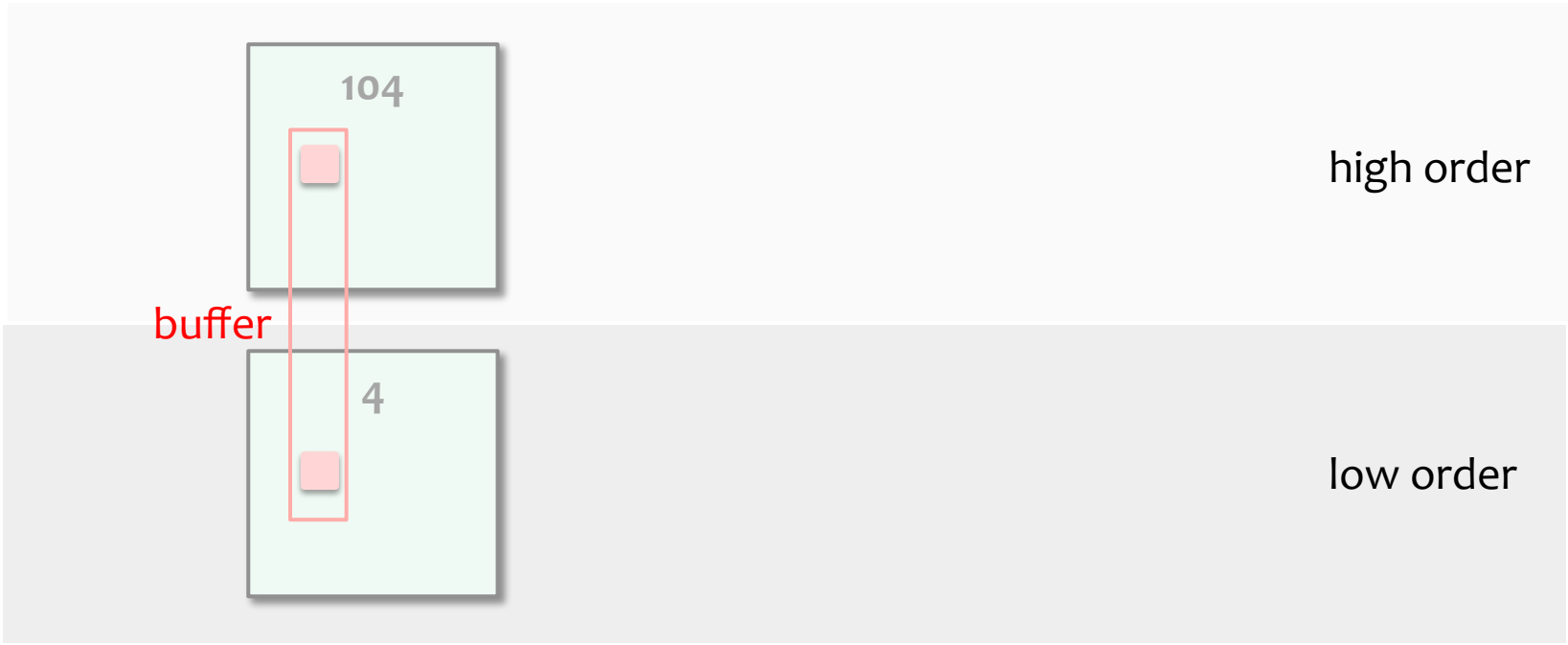
MD5 in CPart

```
typedef pair<int,int> myInt;

vector<myInt>@{[0:64]=(106,6)} k[64];

myInt@(105,5) sumrotate(myInt@(104,4) buffer, ...) {
  myInt@h sum = buffer +@here k[i] + message[g];
  ...
}
```

`buffer` is at `(104,4)`



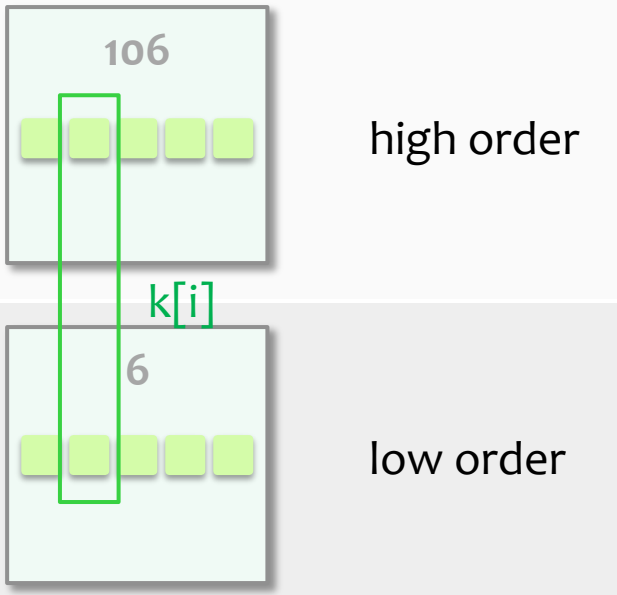
MD5 in CPart

```
typedef pair<int,int> myInt;

vector<myInt>@{[0:64]=(106,6)} k[64];

myInt@(105,5) sumrotate(myInt@(104,4) buffer, ...) {
  myInt@h sum = buffer +@here k[i] + message[g];
  ...
}
```

`k[i]` is at `(106,6)`



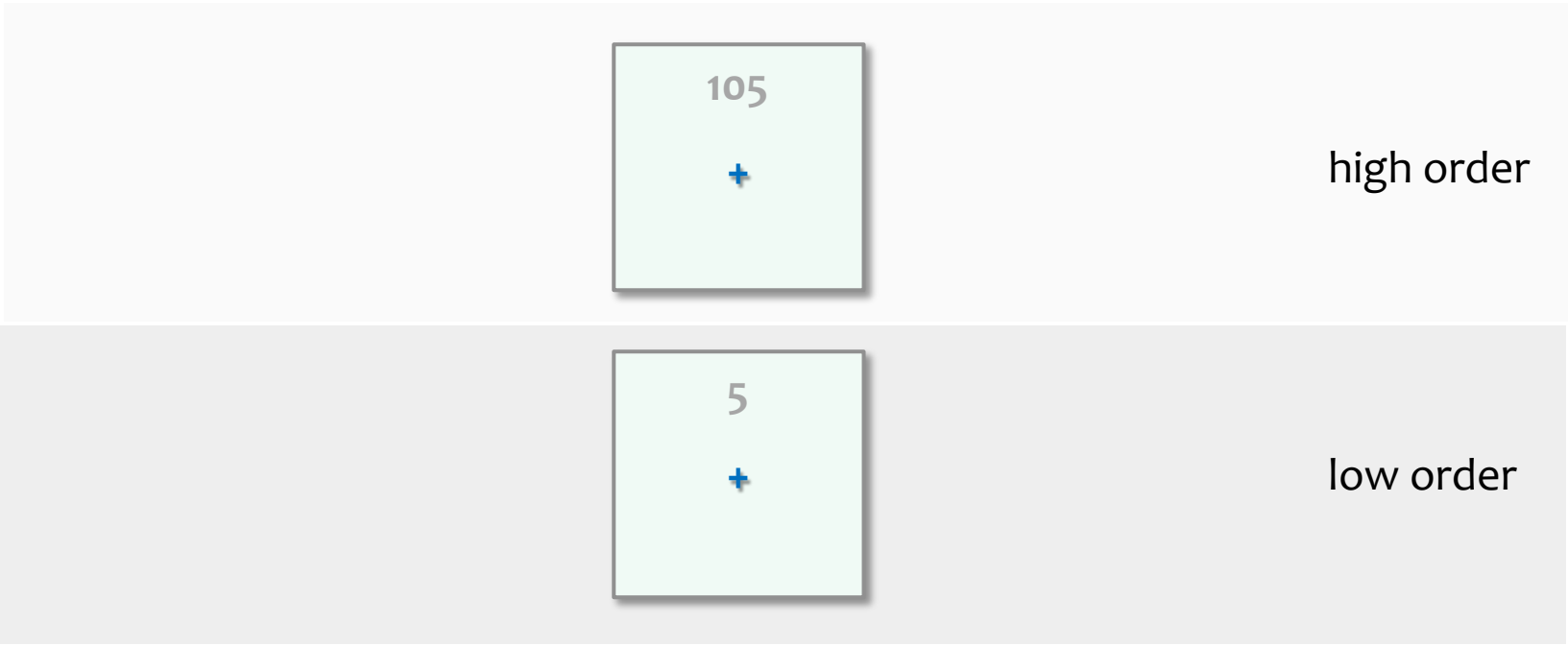
MD5 in CPart

```
typedef pair<int,int> myInt;

vector<myInt>@{[0:64]=(106,6)} k[64];

myInt@(105,5) sumrotate(myInt@(104,4) buffer, ...) {
  myInt@h sum = buffer +@here k[i] + message[g];
  ...
}
```

+	is at (105,5)



MD5 in CPart

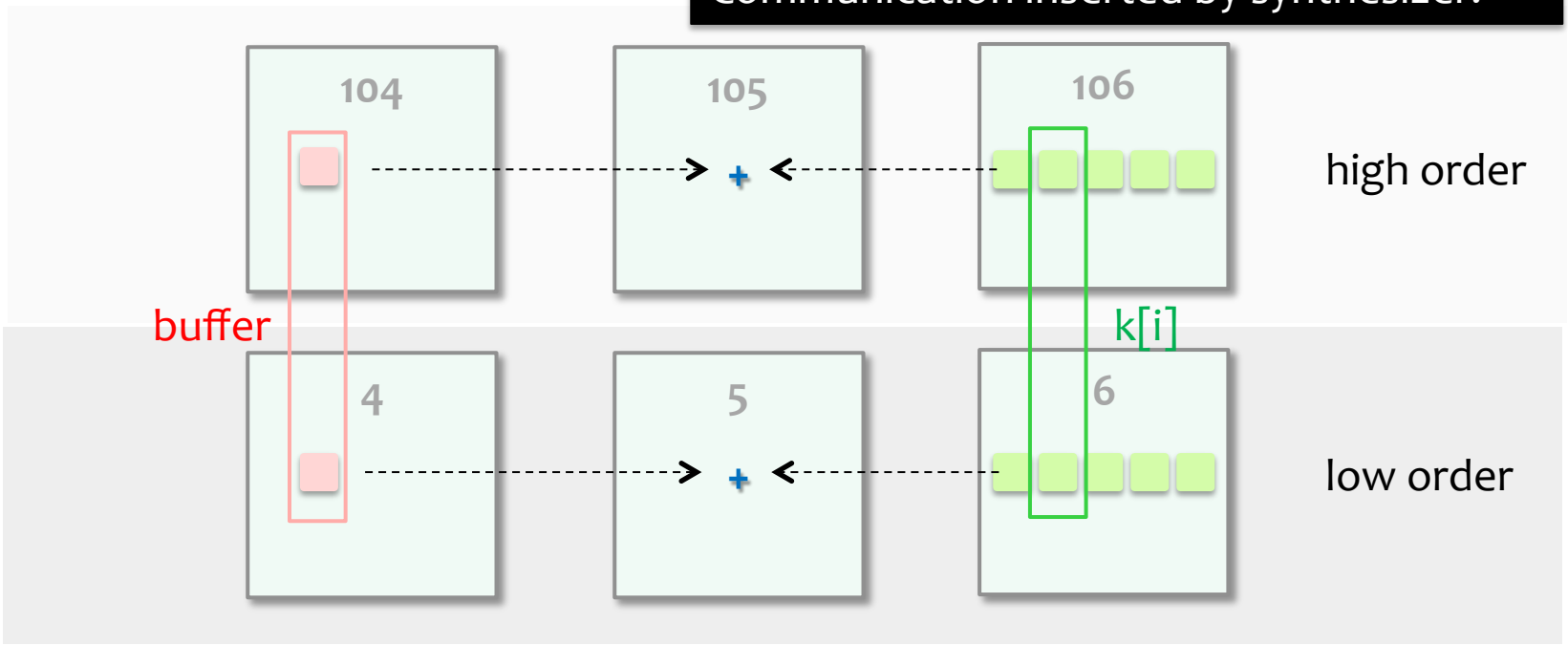
```
typedef pair<int,int> myInt;

vector<myInt>@{[0:64]=(106,6)} k[64];

myInt@(105,5) sumrotate(myInt@(104,4) buffer, ...) {
  myInt@h sum = buffer +@here k[i] + message[g];
  ...
}
```

```
buffer is at (104,4)
+      is at (105,5)
k[i]   is at (106,6)
```

Implicit communication in source program.
Communication inserted by synthesizer.



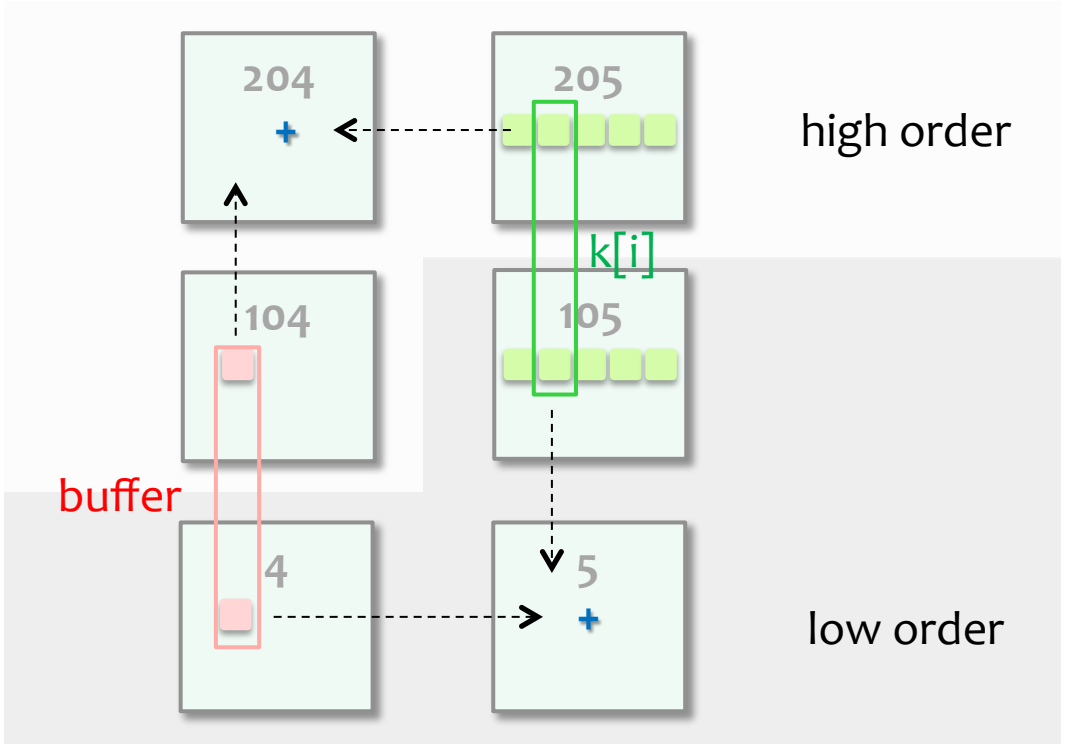
MD5 in CPart

```
typedef pair<int,int> myInt;

vector<myInt>@{[0:64]=(205,105)} k[64];

myInt@(204,5) sumrotate(myInt@(104,4) buffer, ...) {
  myInt@h sum = buffer +@here k[i] + message[g];
  ...
}
```

```
buffer is at (104,4)
+      is at (204,5)
k[i]   is at (205,105)
```



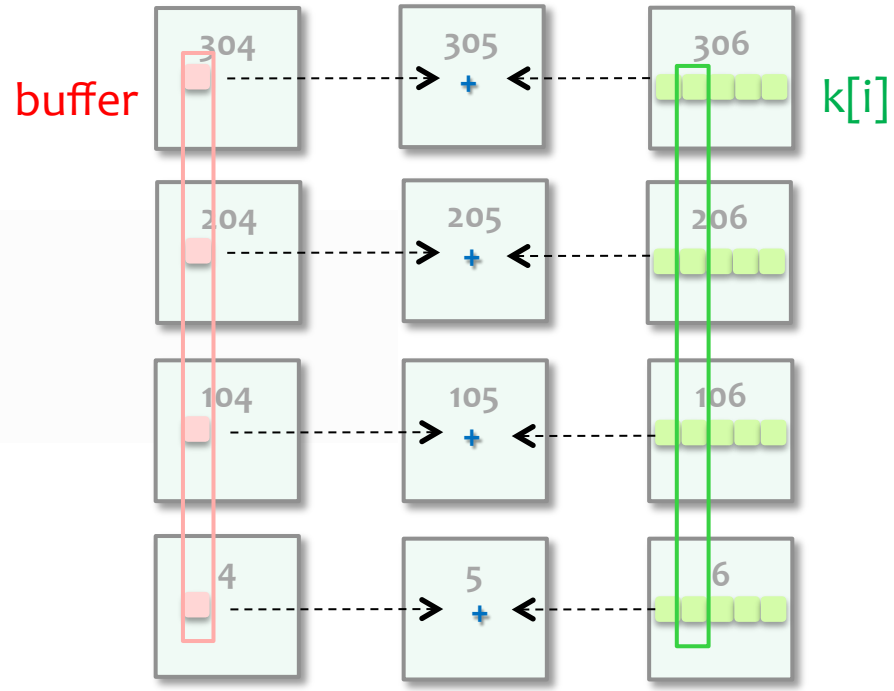
MD5 in CPart

```
typedef vector<int> myInt;

vector<myInt>@{[0:64]={306,206,106,6}} k[64];

myInt@{305,205,105,5} sumrotate(myInt@{304,204,104,4} buffer, ...) {
  myInt@h sum = buffer +@here k[i] + message[g];
  ...
}
```

```
buffer is at {304,204,104,4}
+      is at {305,205,105,5}
k[i]   is at {306,206,106,6}
```



MD5 in CPart

```
typedef pair<int,int> myInt;

vector<myInt>@{[0:64]=(106,6)} k[64];

myInt@(105,5) sumrotate(myInt@(104,4) buffer, ...) {
  myInt@h sum = buffer +@here k[i] + message[g];
  ...
}
```

MD5 in CPart

```
typedef pair<int,int> myInt;

vector<myInt>@{[0:64]=(106,6)} k[64];

myInt@(105,5) sumrotate(myInt@(104,4) buffer, ...) {
  myInt@h sum = buffer +@here k[i] +@?? message[g];
  ...
}
```


+ happens at here which is (105,5)

+ happens at where the synthesizer decides

Summary: Language & Compiler

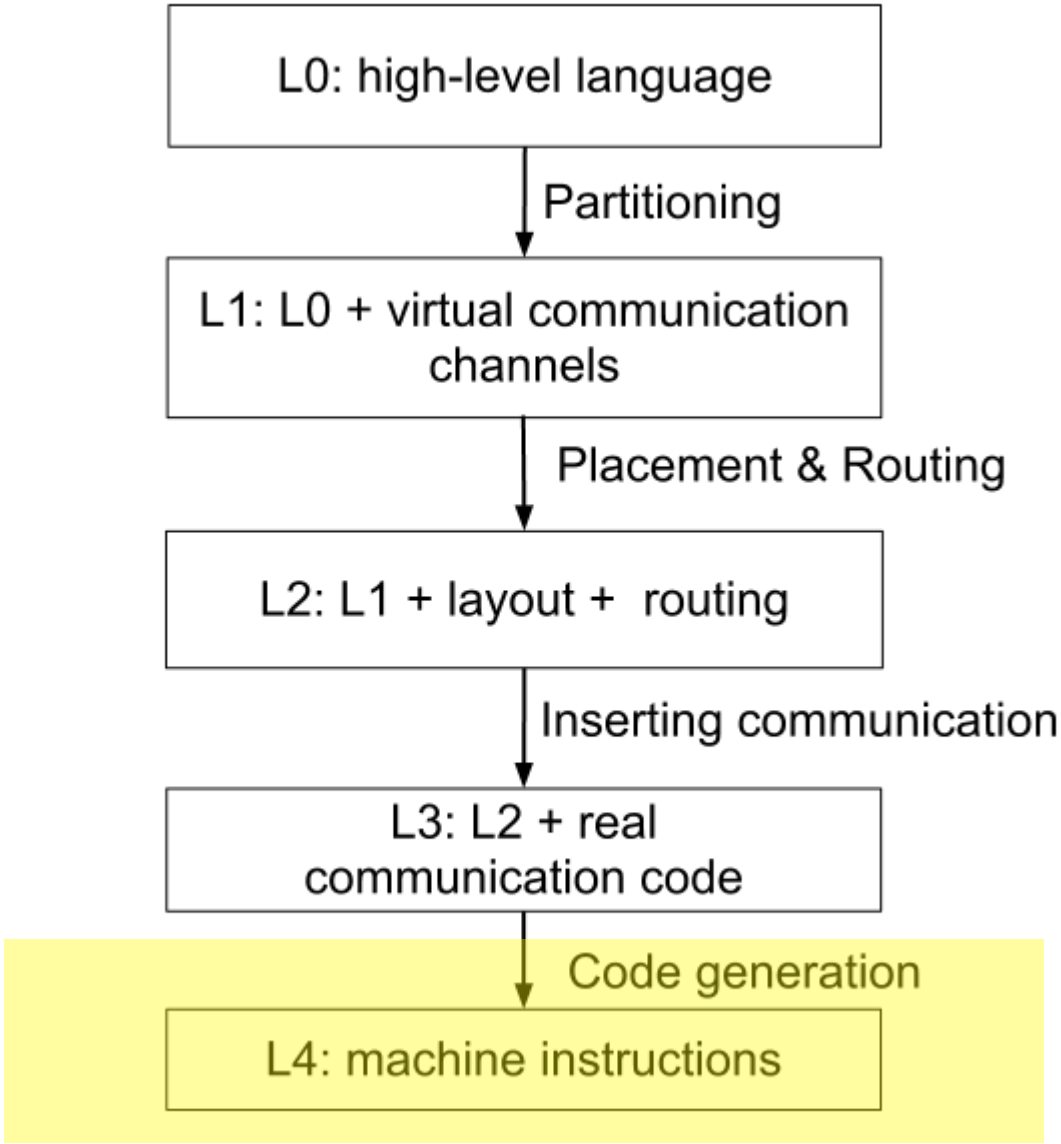
Language features:

- Specify code and data placement by annotation
- No explicit communication required
- Option to not specifying place

Synthesizer fills in holes such that:

- number of messages is minimized
- code can fit in each core

Project Pipeline



Program Synthesis for Code Generation

Synthesizing optimal code

- **Input:** unoptimized code (the spec)
- Search space of all programs

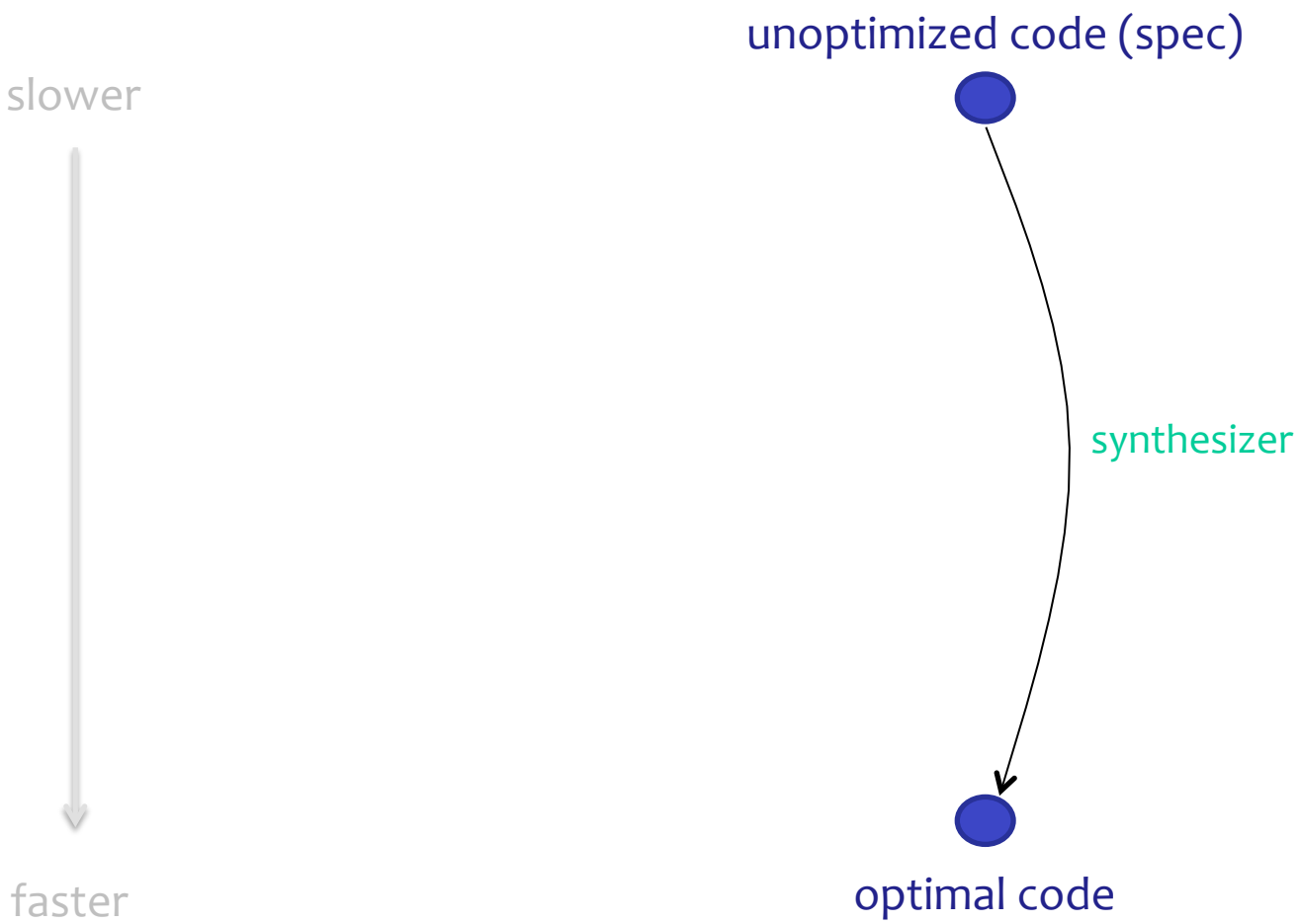
Synthesizing optimal library code

- **Input:** sketch + spec
- Search completions of the sketch

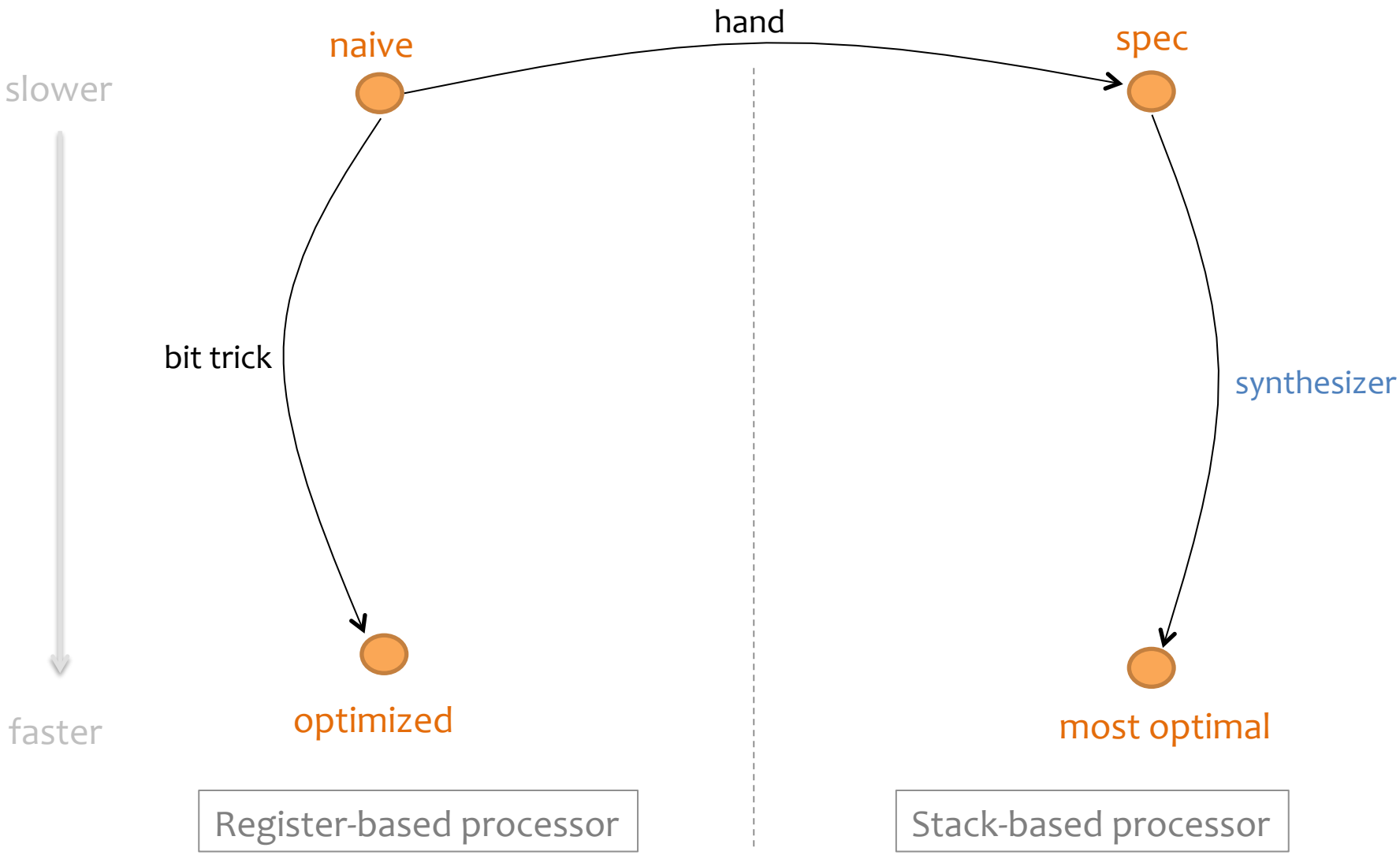
Synthesizing communication code

- **Input:** program with virtual channels
- Insert actual communication code

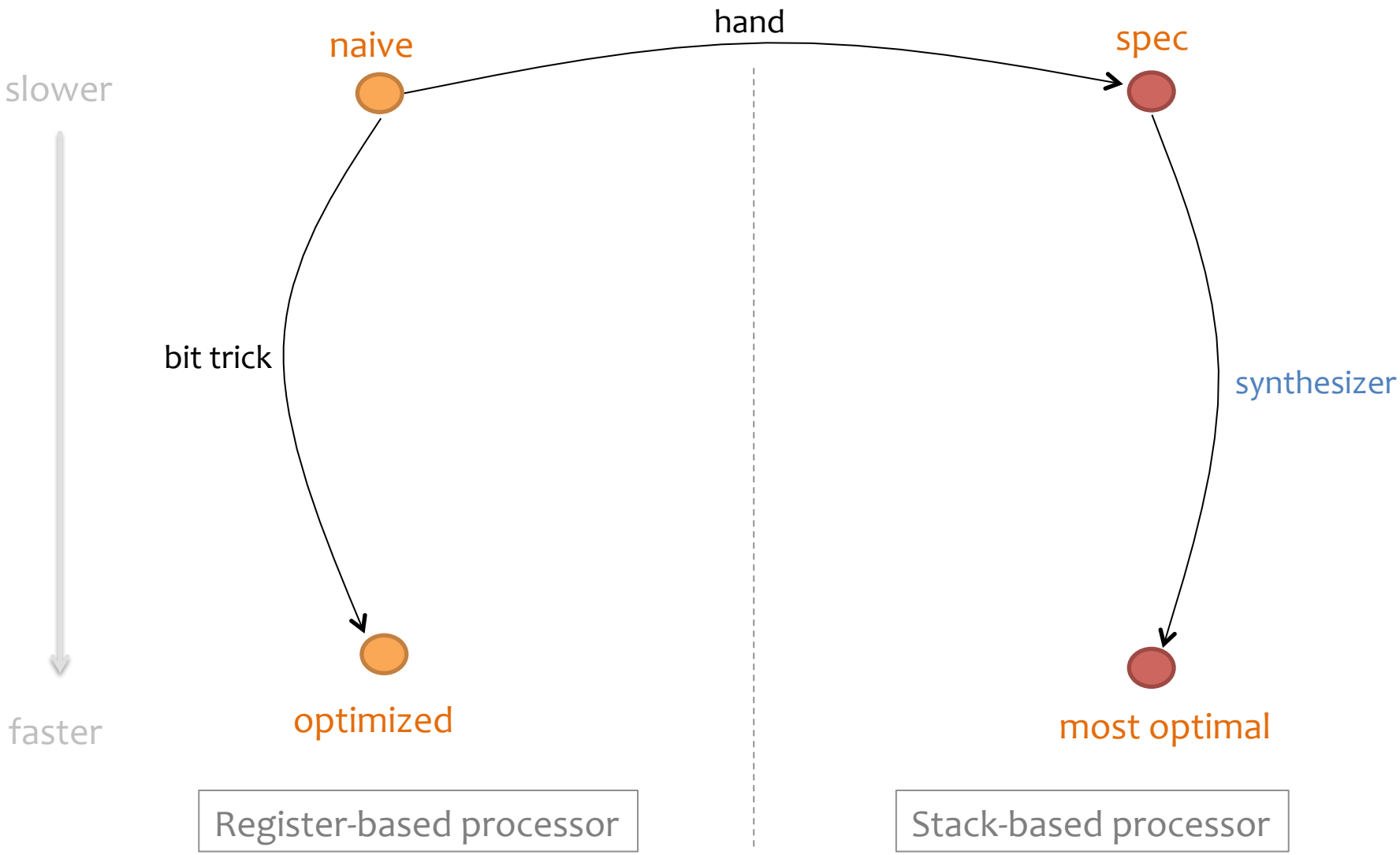
1) Synthesizing optimal code



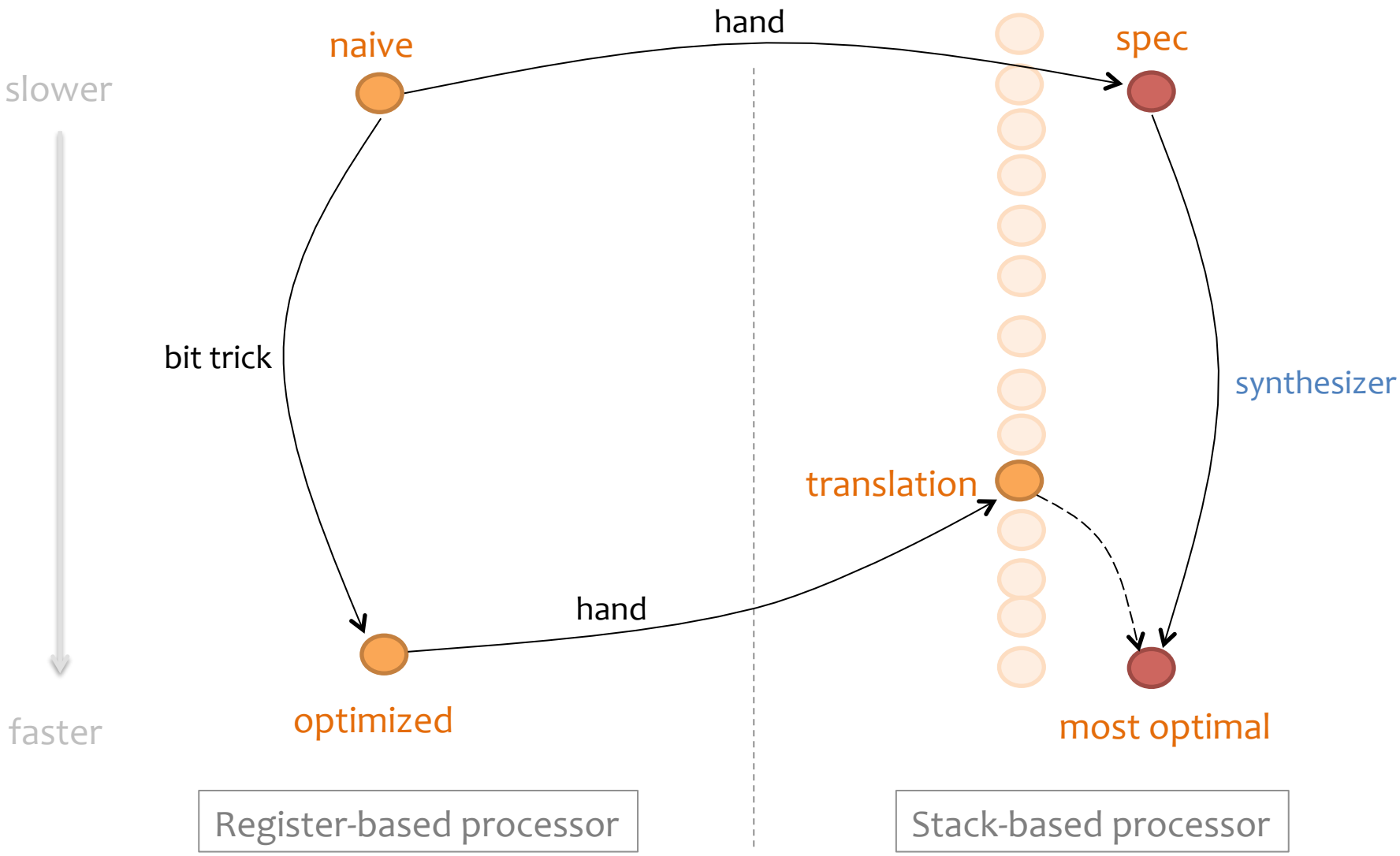
Our Experiment



Our Experiment



Comparison


slower
faster
naive
hand
spec
bit trick
synthesizer
translation
hand
optimized
most optimal
Register-based processor
Stack-based processor

Preliminary Synthesis Times

Synthesizing a program with

8 unknown instructions

takes 5 second to 5 minutes

Synthesizing a program up to

~25 unknown instructions

within 50 minutes

Preliminary Results

Program	Description	Approx. Speedup	Code length reduction
x – (x & y)	Exclude common bits	5.2x	4x
~(x – y)	Negate difference	2.3x	2x
x | y	Inclusive or	1.8x	1.8x
(x + 7) & -8	Round up to multiple of 8	1.7x	1.8x
(x & m) | (y & ~m)	Replace **x** with **y** where bits of **m** are 1's	2x	2x
(y & m) | (x & ~m)	Replace **y** with **x** where bits of **m** are 1's	2.6x	2.6x
x' = (x & m) | (y & ~m) y' = (y & m) | (x & ~m)	Swap **x** and **y** where bits of **m** are 1's	2x	2x

Code Length

Program	Original Length	Output Length
x – (x & y)	8	2
~(x – y)	8	4
x | y	27	15
(x + 7) & -8	9	5
(x & m) | (y & ~m)	22	11
(y & m) | (x & ~m)	21	8
x' = (x & m) | (y & ~m) y' = (y & m) | (x & ~m)	43	21

2) Synthesizing optimal library code

Input:

Sketch: program with holes to be filled

Spec: program in any programing language

Output:

Complete program with filled holes

Example: Integer Division by Constant

Naïve Implementation:

Subtract divisor until reminder < divisor.

# of iterations = output value **Inefficient!**

Better Implementation:

$$\text{quotient} = (M * n) >> s$$

n - input

M - "magic" number

s - shifting value

M and s depend on the number of bits and constant divisor.

Example: Integer Division by 3

Spec:

$$n/3$$

Sketch:

```
quotient = (?? * n) >> ??
```

Preliminary Results

Program	Solution	Synthesis Time (s)	Verification Time (s)	# of Pairs
x/3	(43691 * x) >> 17	2.3	7.6	4
x/5	(209716 * x) >> 20	3	8.6	6
x/6	(43691 * x) >> 18	3.3	6.6	6
x/7	(149797 * x) >> 20	2	5.5	3
deBruijn: $Log_2 x$ (x is power of 2)	deBruijn = 46, Table = {7, 0, 1, 3, 6, 2, 5, 4}	3.8	N/A	8

Note: these programs work for 18-bit number except Log2x is for 8-bit number.

3) Communication Code for GreenArray

Synthesize communication code between nodes

Interleave communication code with computational code such that

- There is no deadlock.
- The runtime or energy comsumption of the synthesized program is minimized.

The key to the success of
low-power computing lies in
inventing better and more effective
programming frameworks.