



Ascenium: A Continuously Reconfigurable Architecture

Robert Mykland

Founder/CTO

robert@ascenium.com

August, 2005

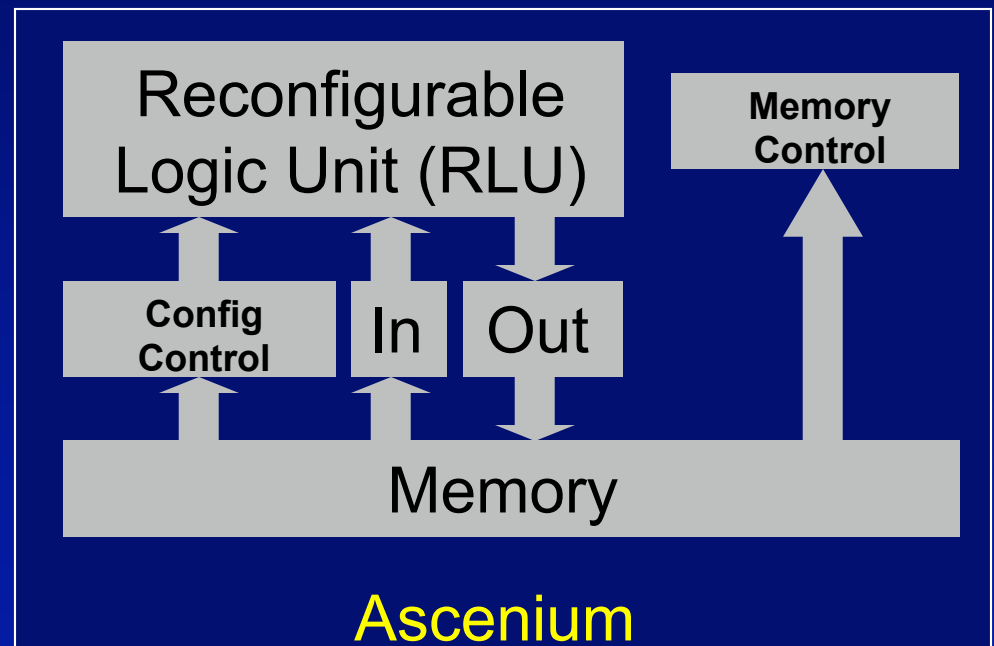
Ascenium: A Continuously Reconfigurable Processor

- **Continuously reconfigurable approach provides:**
 - The computational efficiency of direct logic implementations in ASICs
 - All the flexibility of microprocessors (pure software)
- **Unique architecture enables the array to be effectively targeted by an ANSI Standard C compiler**
 - CPU logic is continuously redefined to match the compiler's needs for each instruction (≈ 20 cycles)
- **Massive parallelism even in programs with little or no inherent instruction-level parallelism**
 - Dozens of lines of sequential C code are executed every instruction



Ascenium Block Diagram

- ◆ No instruction set
- ◆ RLU is combinatorial logic that settles
- ◆ Memory is accessed in statically scheduled block operations
- ◆ Two instruction streams
 - 1) Memory
 - 2) Configuration
- ◆ Whole loops are often executed as one RLU instruction

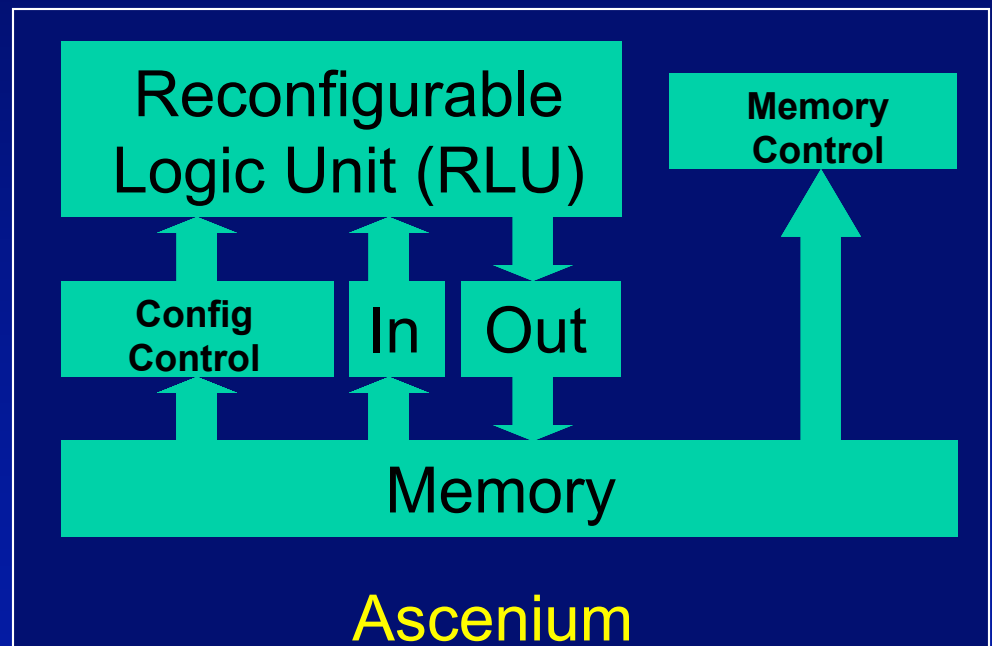
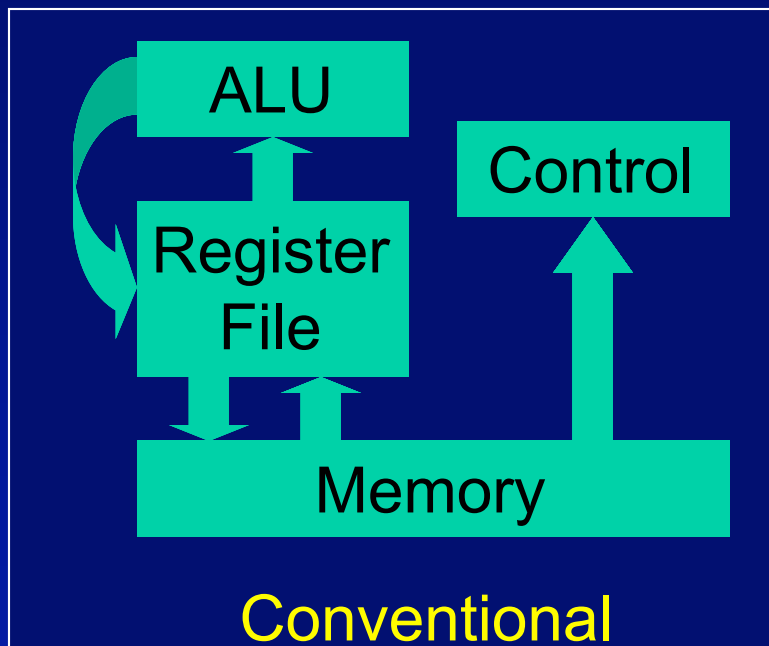


Ascenium Basic Operation

12

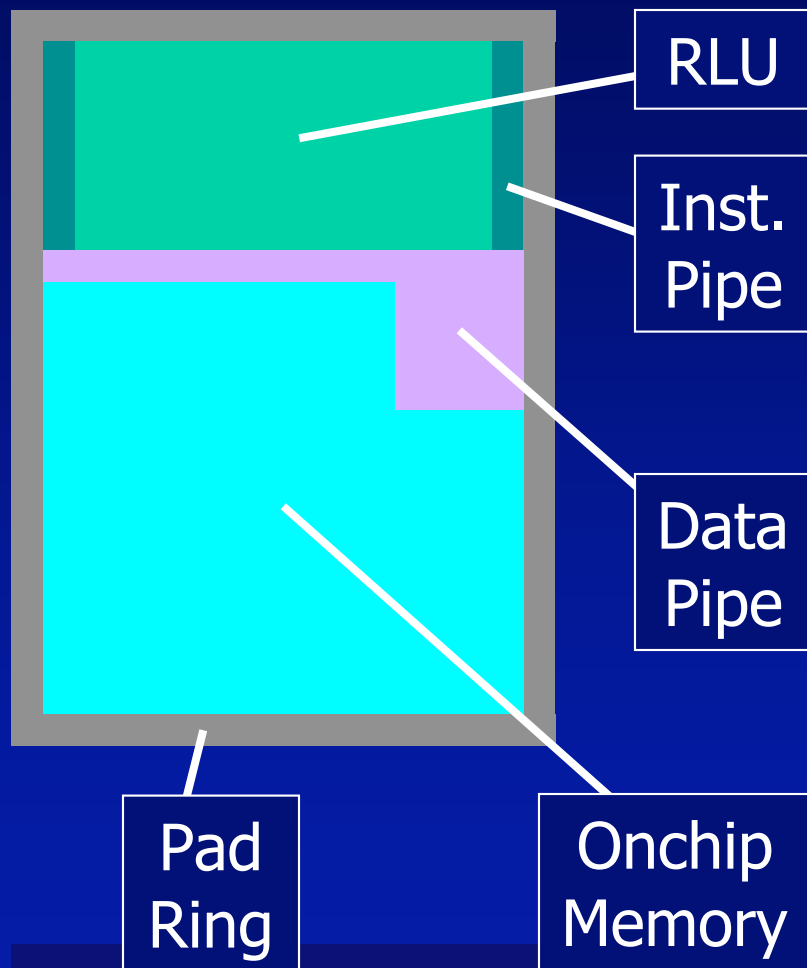
RISC Instructions

165



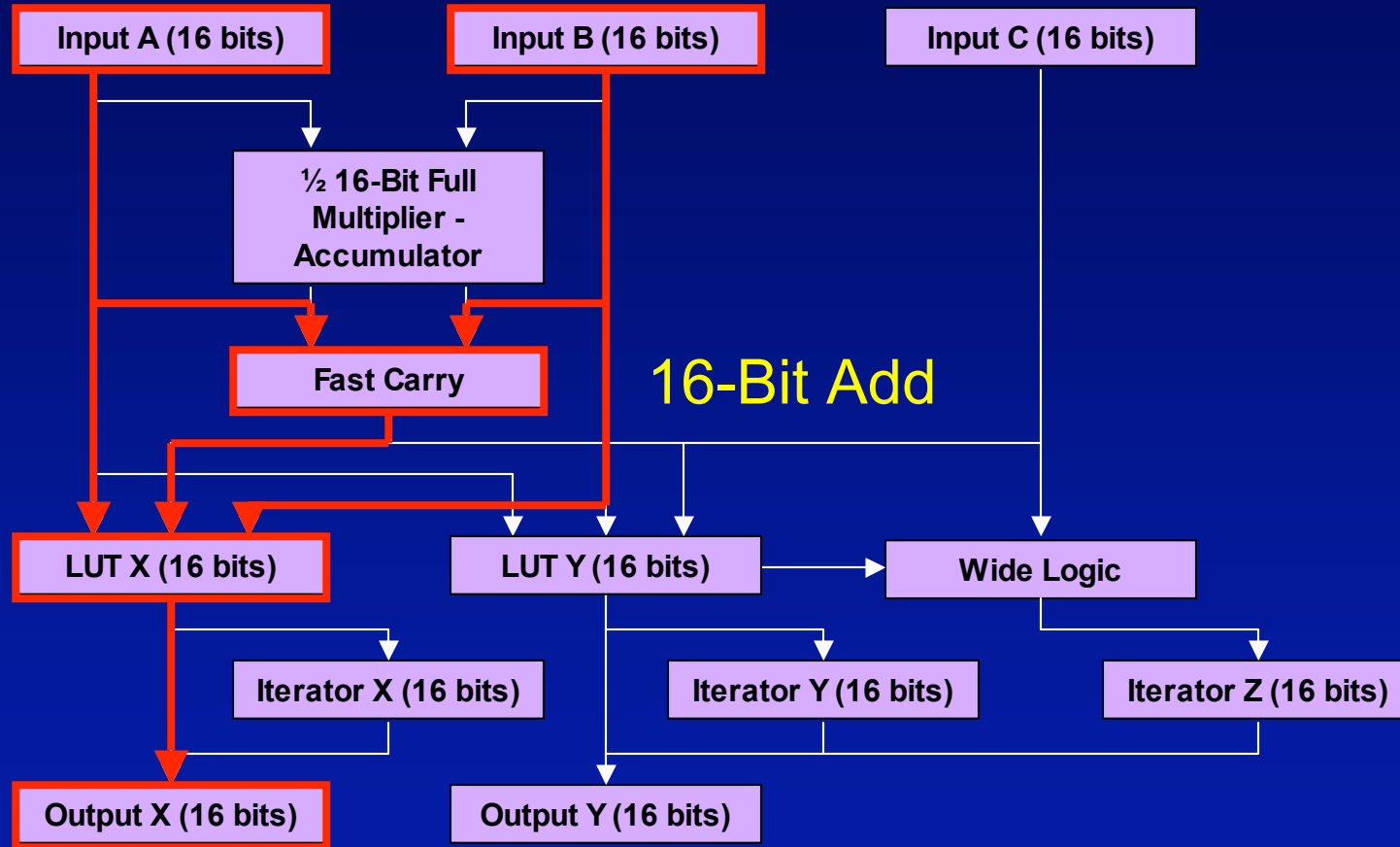
Clock = 24

A128X16 Rough Layout

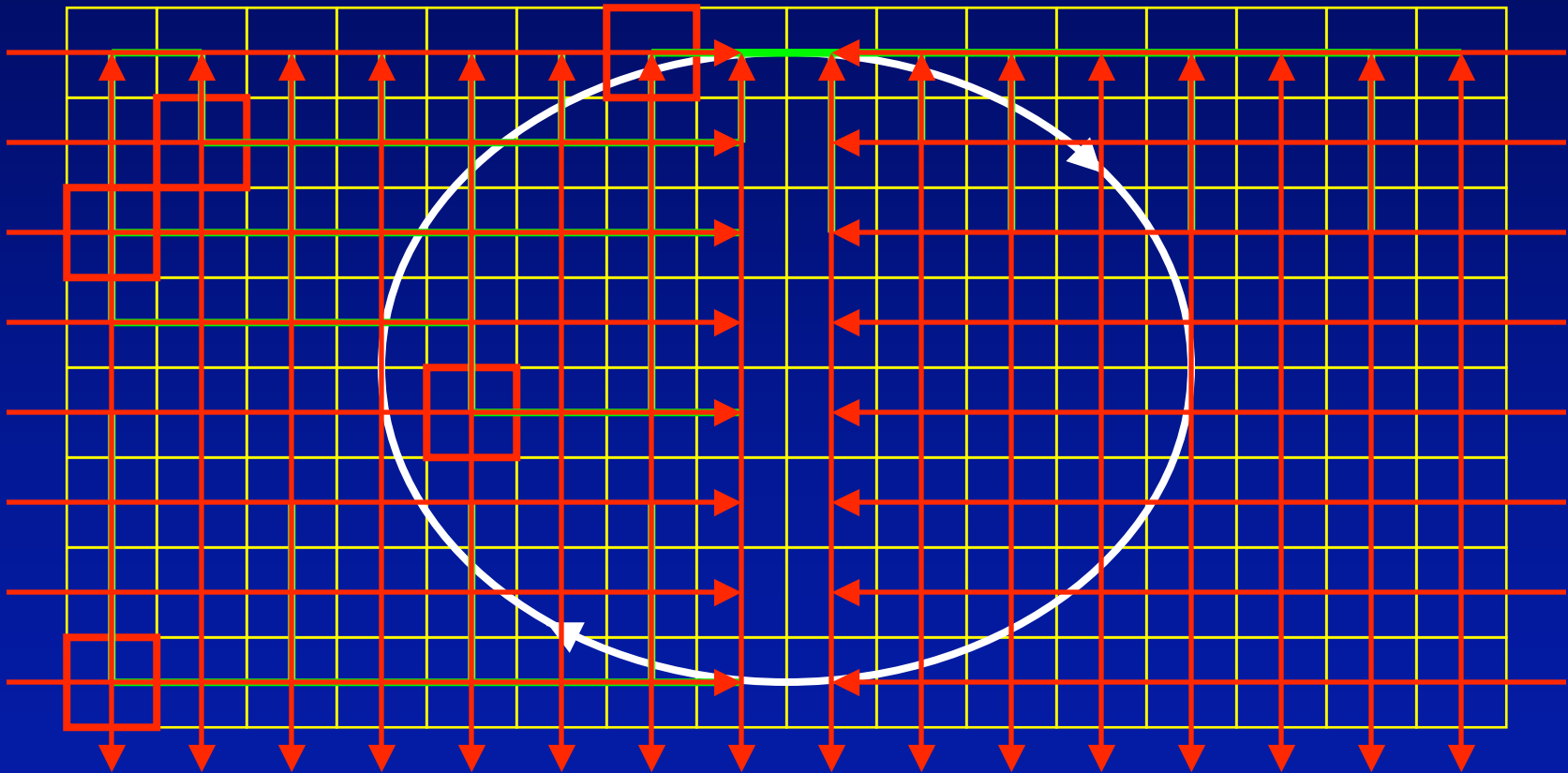


- 130 nm process
- 128 16-bit logic elements
- 64 MACs/clock cycle
- 200 MHz DDR II = 3.2 GB/s external memory bandwidth
- 256KB onchip memory
- 19.2 GB/s peak onchip memory bandwidth
- 500 mW @ 200 MHz
- 52 mm²

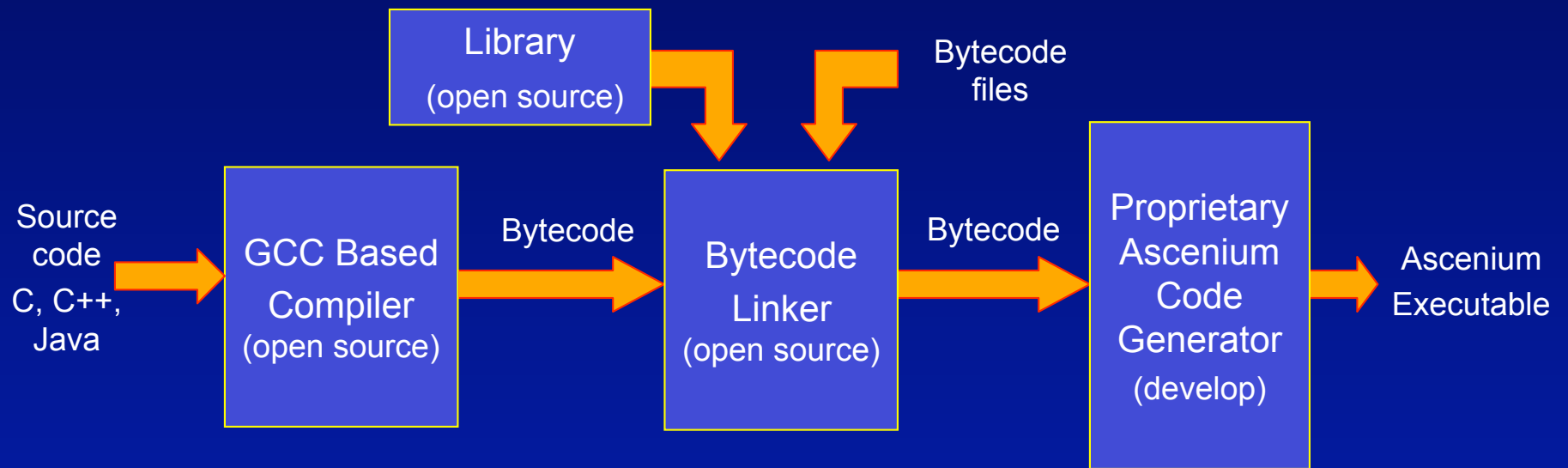
Logic Element



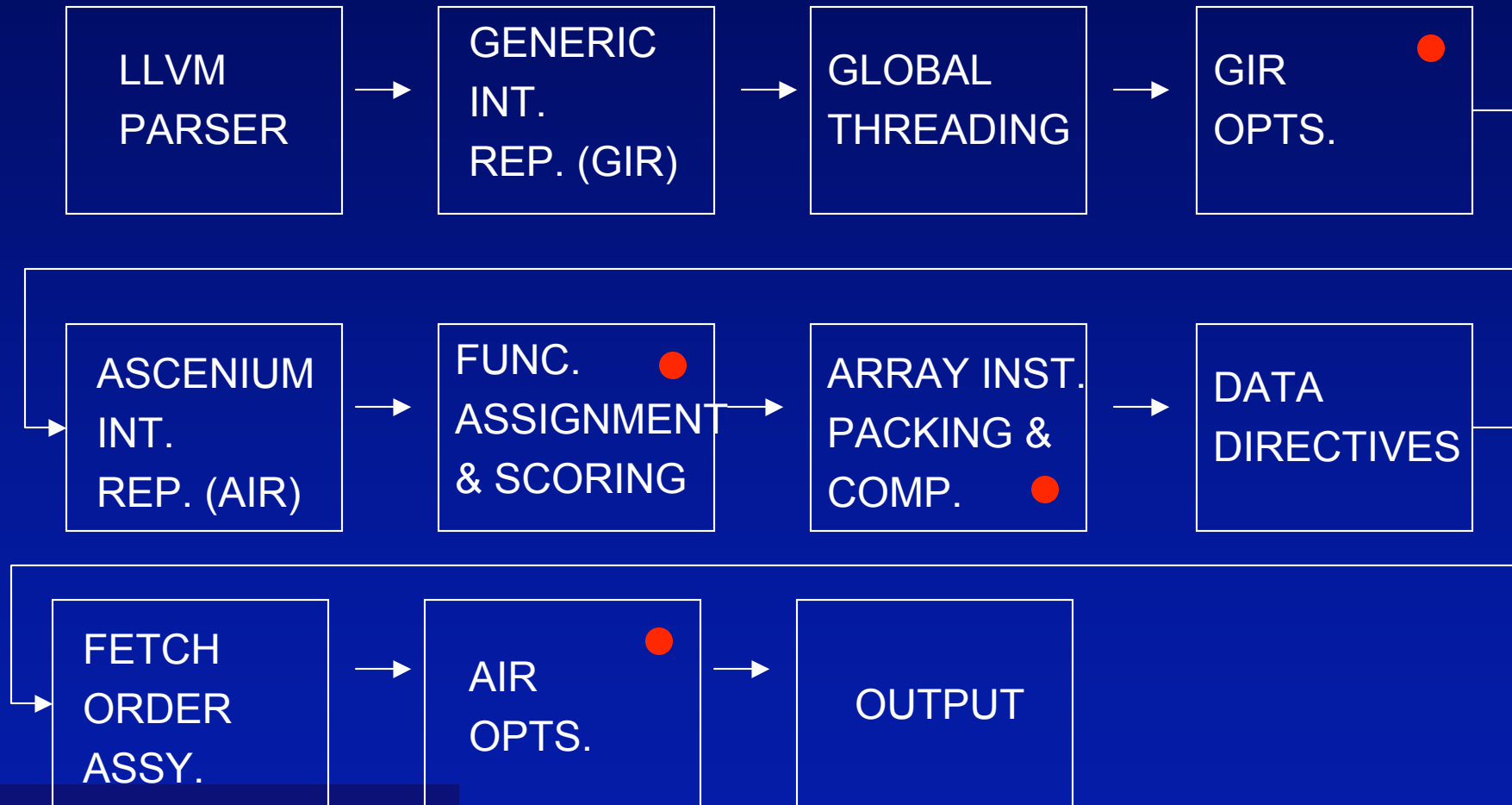
Array Connectivity



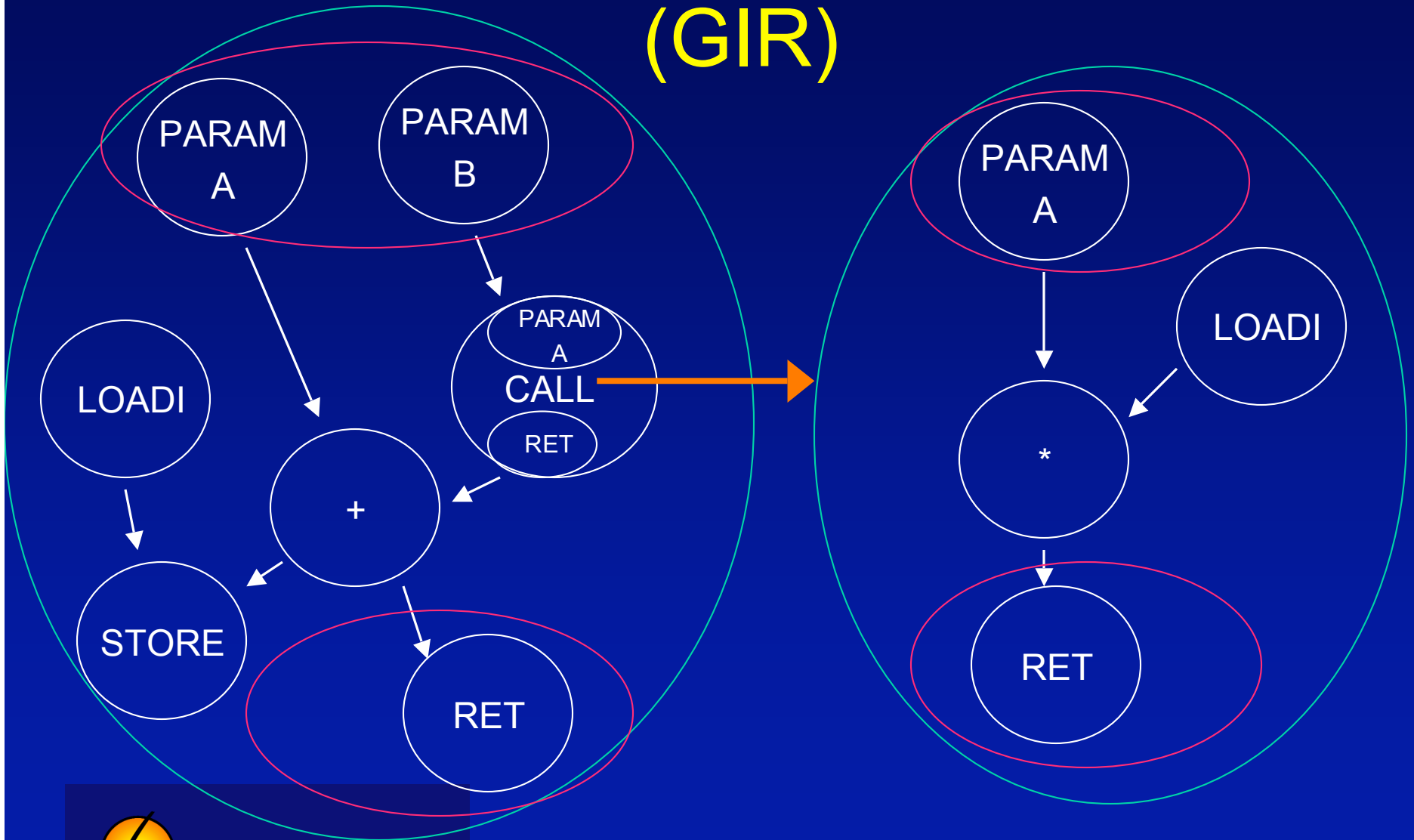
Compiler Block Diagram



Code Generator Diagram



Generic Intermediate Representation (GIR)



FFT Inner Loop C Code

```
i1 = i0 + n2;  
i2 = i1 + n2;  
i3 = i2 + n2;  
r1 = x[2 * i0] + x[2 * i2];  
r2 = x[2 * i0] - x[2 * i2];  
t = x[2 * i1] + x[2 * i3];  
x[2 * i0] = r1 + t;  
r1 = r1 - t;  
s1 = x[2 * i0 + 1] + x[2 * i2 + 1];  
s2 = x[2 * i0 + 1] - x[2 * i2 + 1];  
t = x[2 * i1 + 1] + x[2 * i3 + 1];  
x[2 * i0 + 1] = s1 + t;  
s1 = s1 - t;
```

```
x[2 * i2] = (r1 * co2 + s1 * si2) >>15;  
x[2 * i2 + 1] = (s1 * co2 - r1 * si2) >>15;  
t = x[2 * i1 + 1] - x[2 * i3 + 1];  
r1 = r2 + t;  
r2 = r2 - t;  
t = x[2 * i1] - x[2 * i3];  
s1 = s2 - t;  
s2 = s2 + t;  
x[2 * i1] = (r1 * co1 + s1 * si1) >>15;  
x[2 * i1 + 1] = (s1 * co1 - r1 * si1) >>15;  
x[2 * i3] = (r2 * co3 + s2 * si3) >>15;  
x[2 * i3 + 1] = (s2 * co3 - r2 * si3) >>15;
```

FFT Inner Loop Virtual Circuit



FFT Loop Array Instruction

					MM			A	D	C	F	I	J	GG	HH
				NN	KK			B	E	H	G	K	M	L	N
			EE		FF		JJ		P		Q		T		U
		W		Z				O		R		S		V	
			X		Y	LL	II		AA		BB		CC		DD
	AA		BB		CC		DD				X		Y	LL	II
O		R		S		V				W		Z			
	P		Q		T		U				EE		FF		JJ
B	E	H	G	K	M	L	N						NN	KK	
A	D	C	F	I	J	GG	HH							MM	

X0 Y0 X1 Y1 C1 S1 X0 Y0 X0 Y0 X1 Y1 C1 S1 X0 Y0
 X2 Y2 X3 Y3 Y3 X1 Y1 X2 X2 Y2 X3 Y3 Y3 X1 Y1 X2
 C2 S2 C3 S3 0 X3 0 Y2 C2 S2 C3 S3 0 X3 0 Y2
 0 0 0 0

FFT Loop Data Directives

1. Fetch the array instruction and data directives
2. Load the array instruction
3. Read $x[t]$ in four 64-bit wide banks 32 times
4. Write $x[t]$ in one 256-bit wide bank 32 times, changing banks every 8 operations
5. Repeat steps 3 and 4 three more times
6. At the same time as 3 –5, read in w coefficients
7. Pass 1: w every clock; 2: w every 4 clocks, then every 16, then every 64
8. Stop

DSP Benchmark Results (Simulated, Hand-Coded)

Benchmark	A128X16 core, 130nm, 250MHz, 250mW	TMS320C64x core, 130nm, 300MHz*, 250mW
256-Point Radix-4 FFT	512 ns	8,920 ns
Real Block FIR Filter	512 ns	6,827 ns
Complex Block FIR Filter	512 ns	6,827 ns
DCT x 24	512 ns	6,080 ns
IIR Filter NR = 2,048	292 ns	6,827 ns

* The 130nm C64x core will operate up to 720 MHz, but burns more power at higher frequencies.



Ascenium Status

- Patents filed
- Chip architecture complete, polishing ongoing
- Prototype compiler complete & demo-able
- Development plan in place
- Raising seed round



Summary

- **Great performance and performance/mW:**
 - >10x programmable DSPs
 - >4x performance/mW of FPGAs
- **Great time-to-market:**
 - Normal GPP software development cycle
- **Great ease-of-use:**
 - Programmers writing standard, non-optimized code in C/C++ get all the performance