Transcomputation

Dr James Anderson FBCS CITP CSci
安德生

Agenda

- Total hardware
- Total software

Puzzle

- How would you arrange a total ALU?
- How would you arrange total memory addressing?
- How would you arrange total I/O?
- Physical errors are, ultimately, unavoidable so what are the consequences of an error in a total system?
- How would you arrange a total CPU?

Definitions

- Totallity - a total function gives a result in its output class when applied to any arguments in its input class
- Waiting time - the time an algorithm takes to complete

Pipeline machines

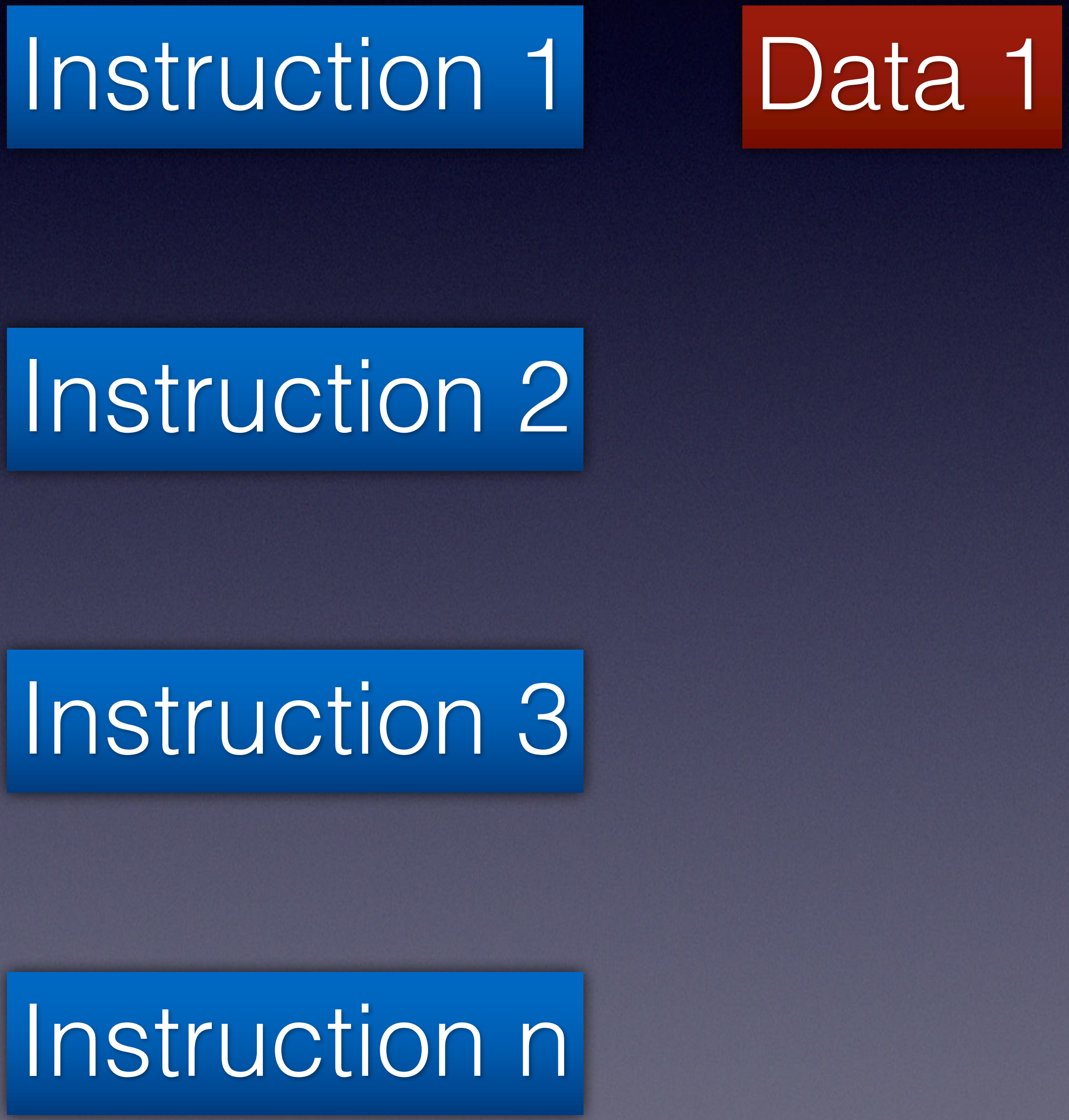
- Totallity + known waiting time = ideal pipeline concurrency
- Pipelines deliver energy efficiency and unlimited scalability with constant computational efficiency

Concurrency

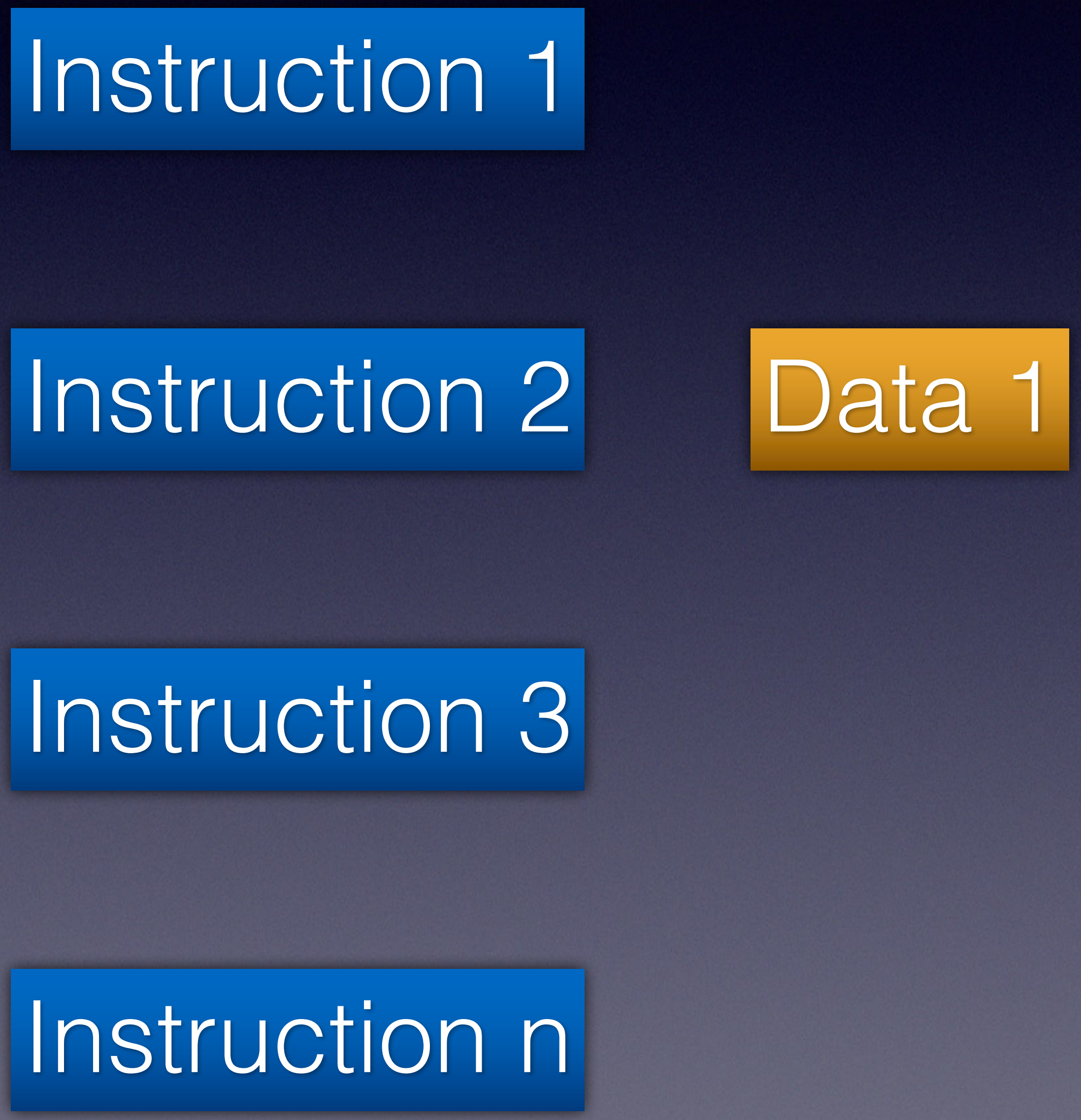
Concurrency

- Parallelism minimises latency
- Pipelining maximises concurrency

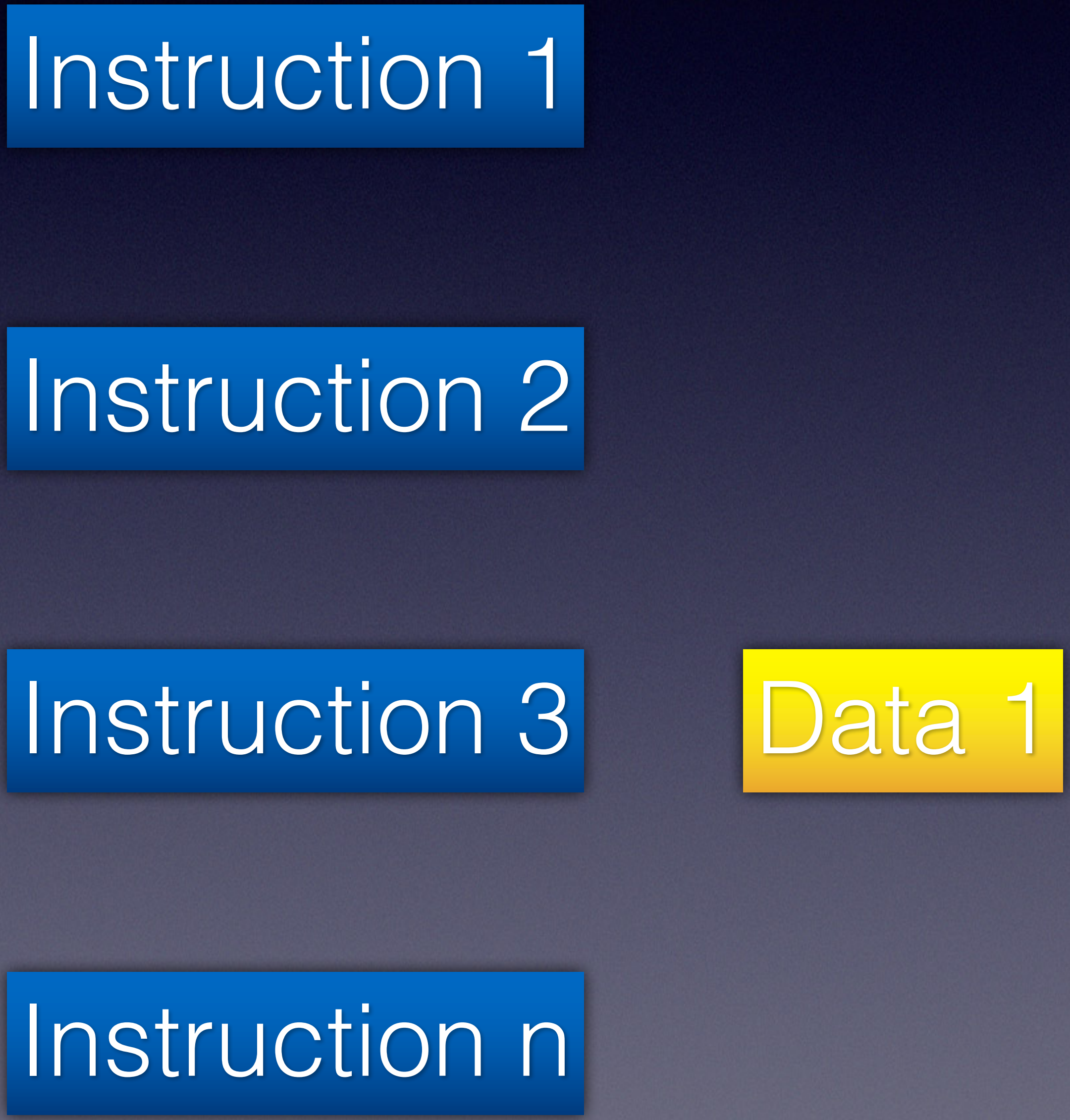
Von Neumann program



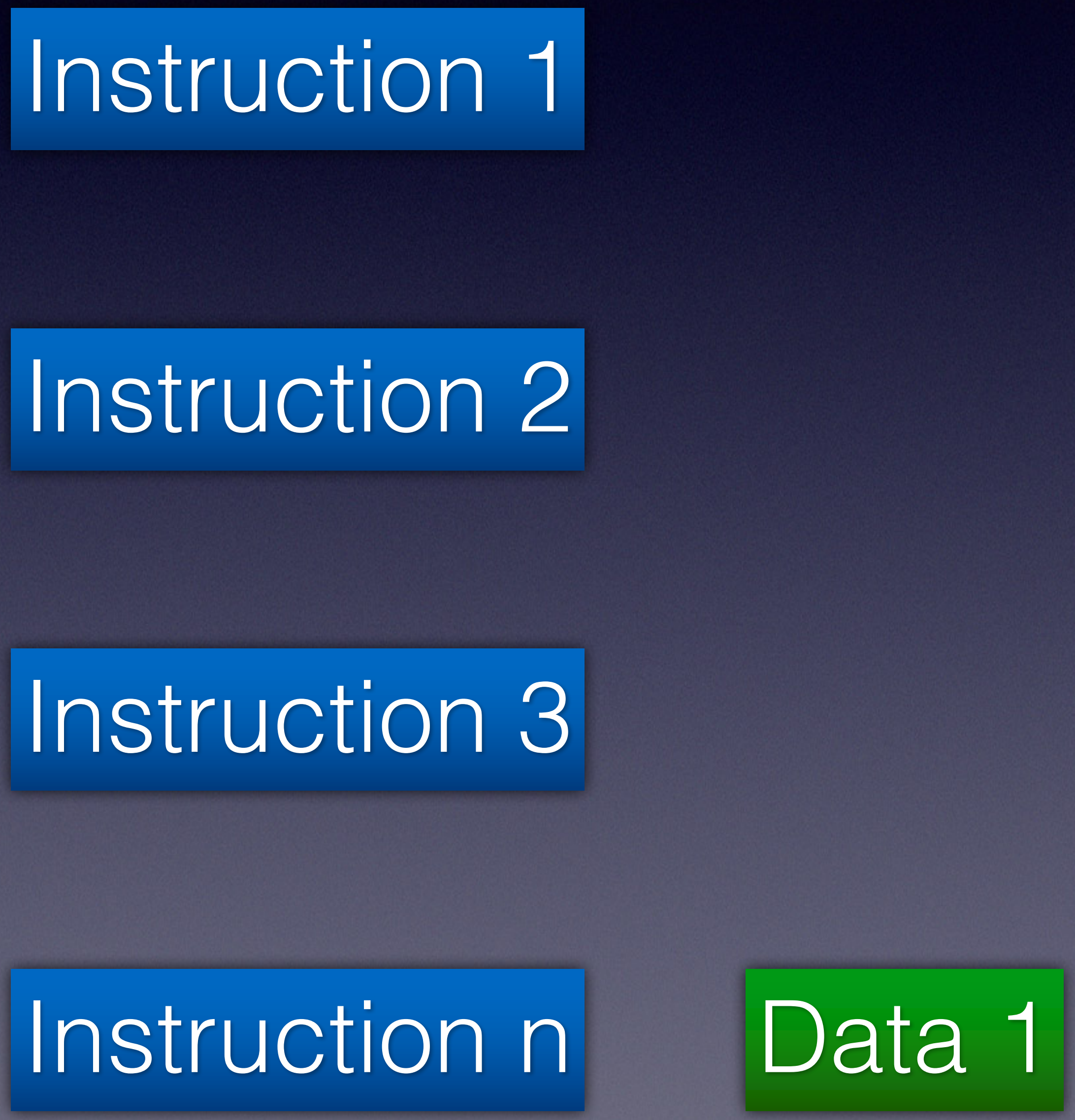
Von Neumann program



Von Neumann program



Von Neumann program



Pipeline program

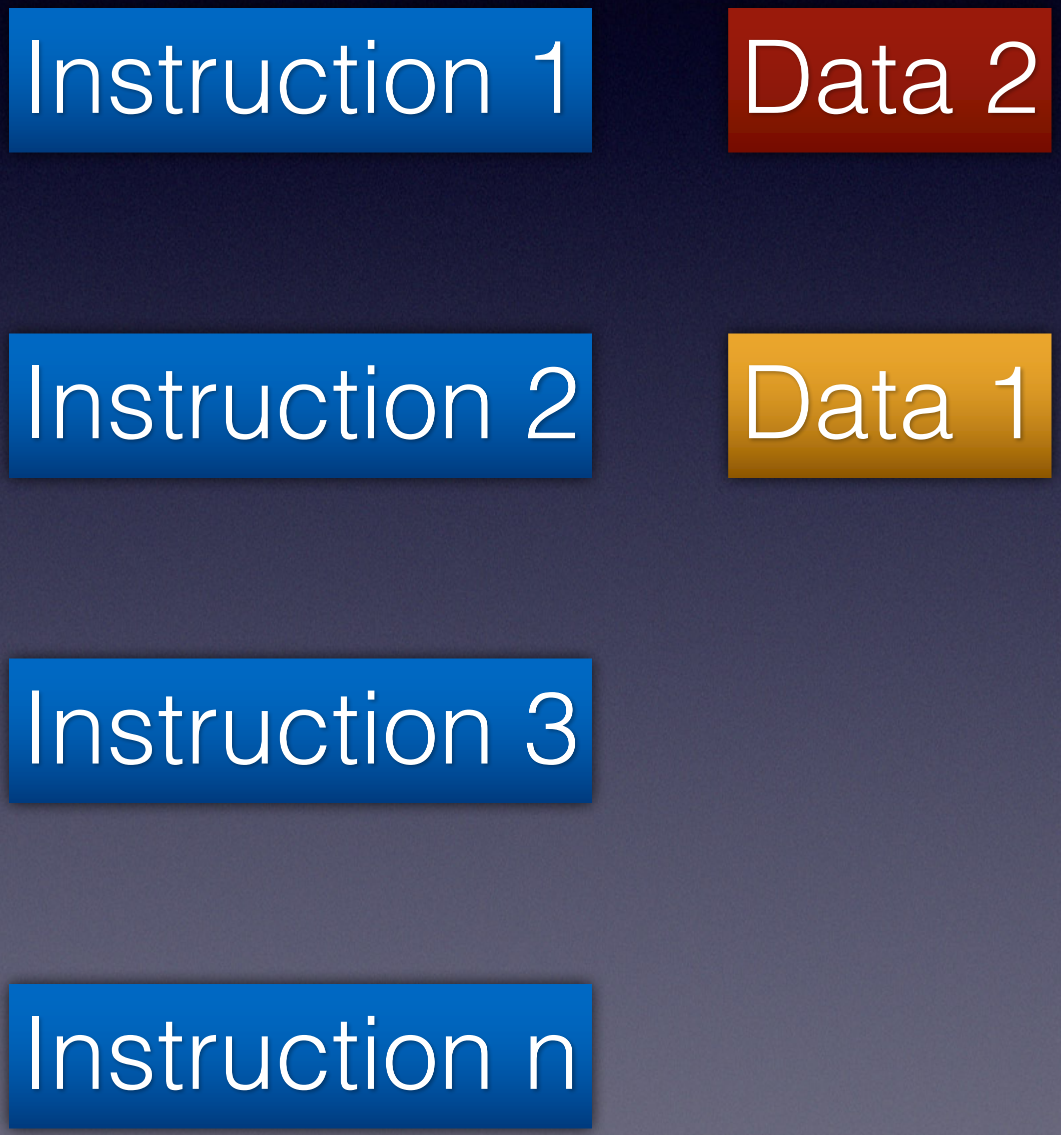
Instruction 1 Data 1

Instruction 2

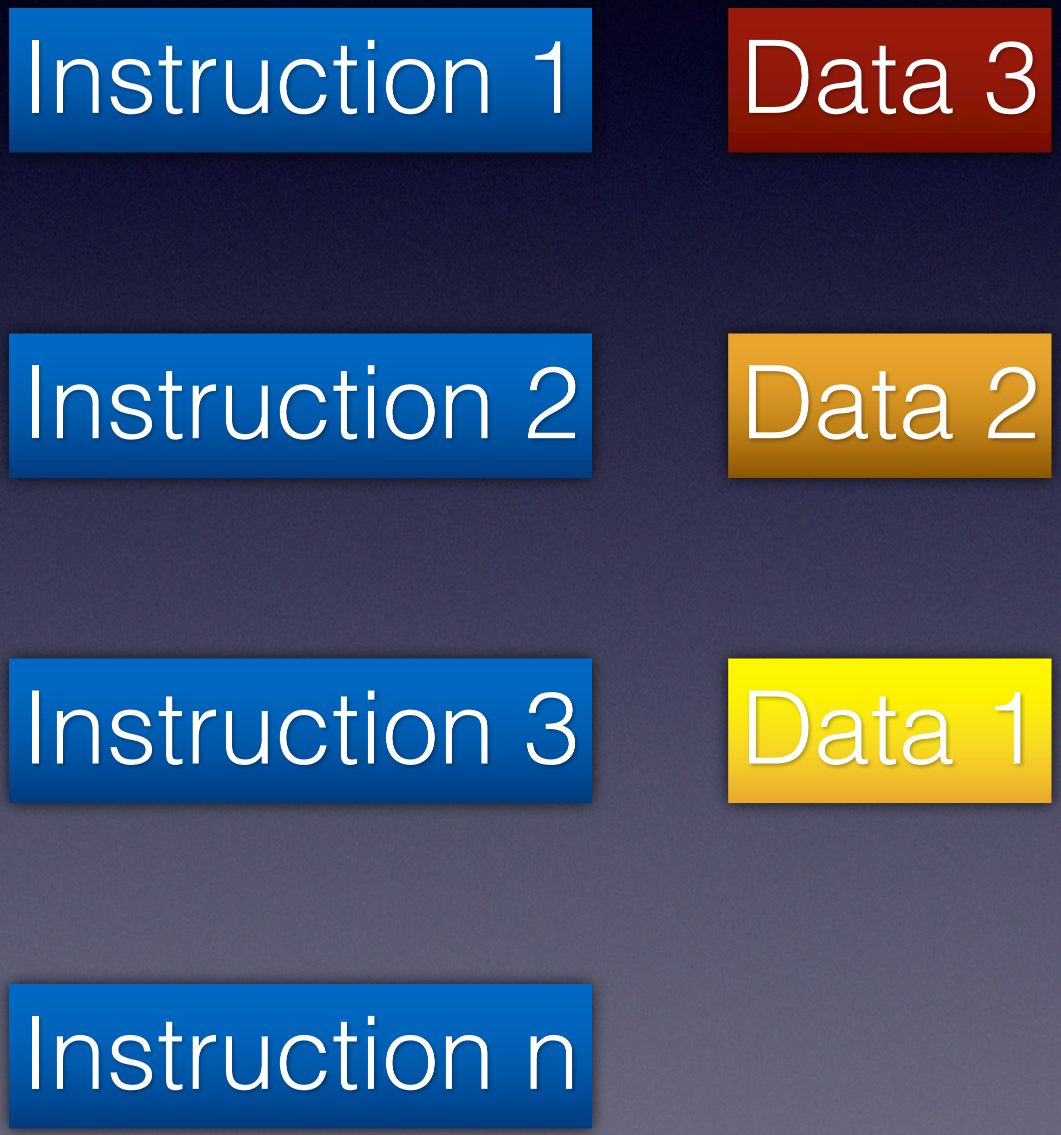
Instruction 3

Instruction n

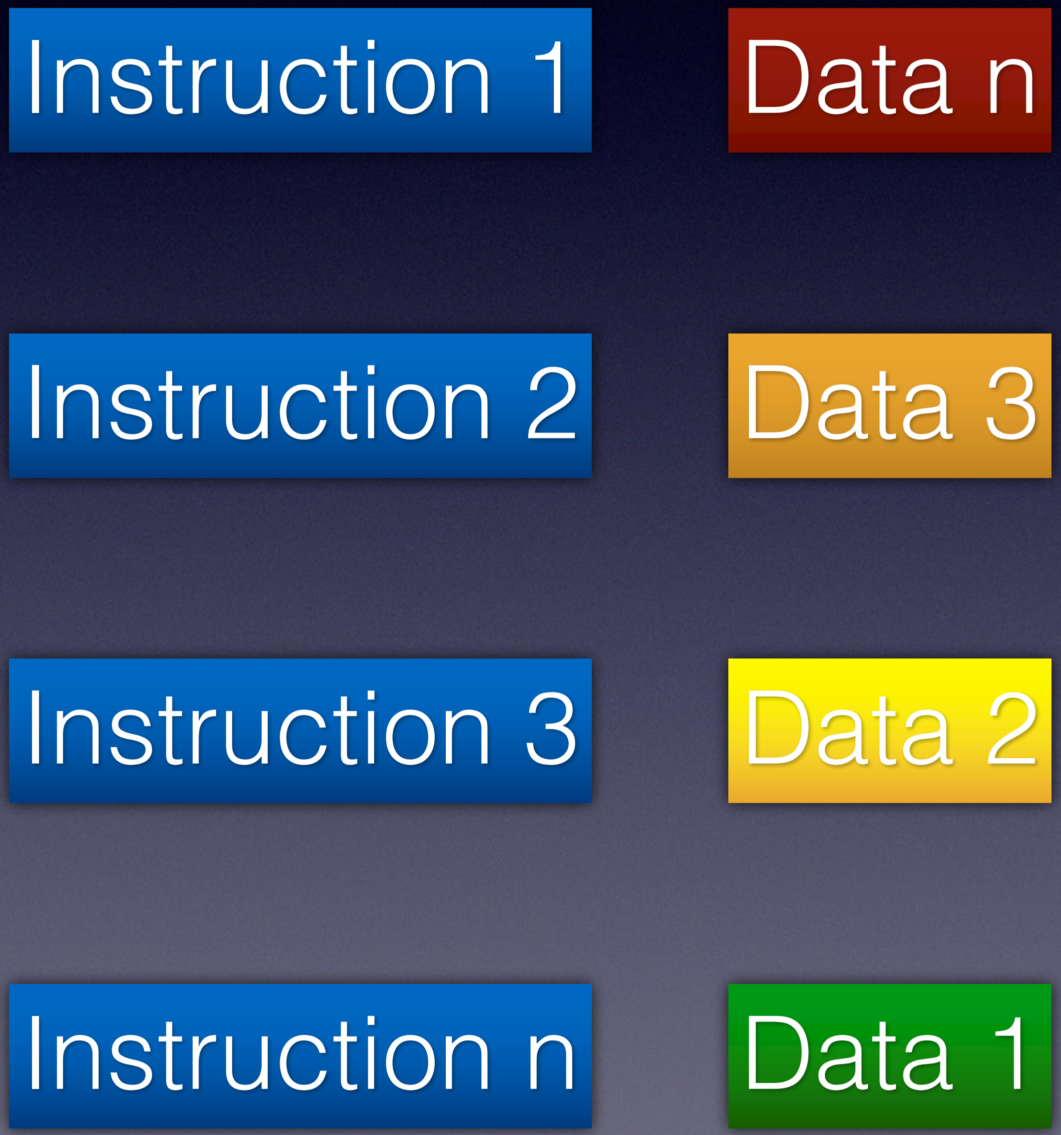
Pipeline program



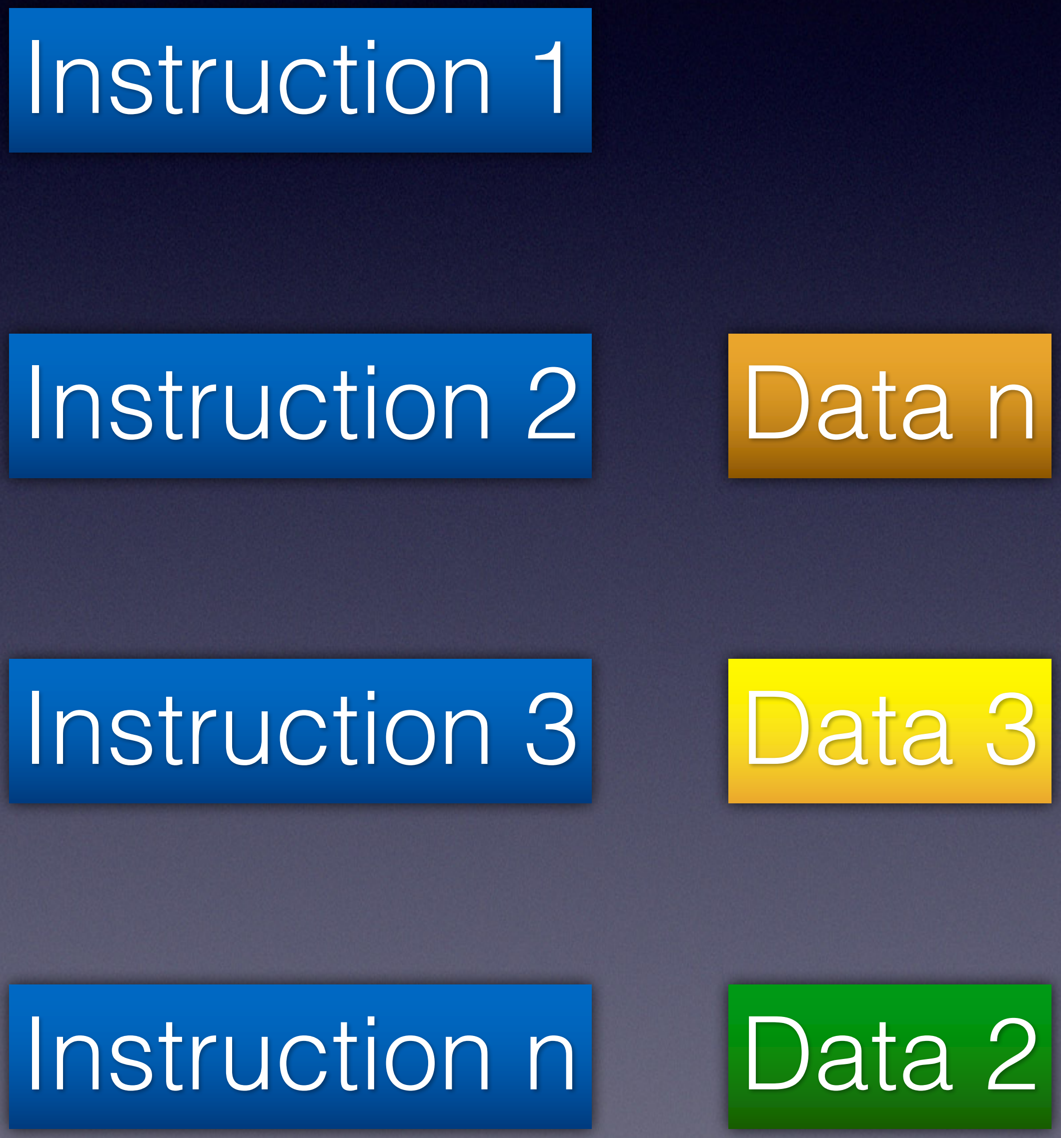
Pipeline program



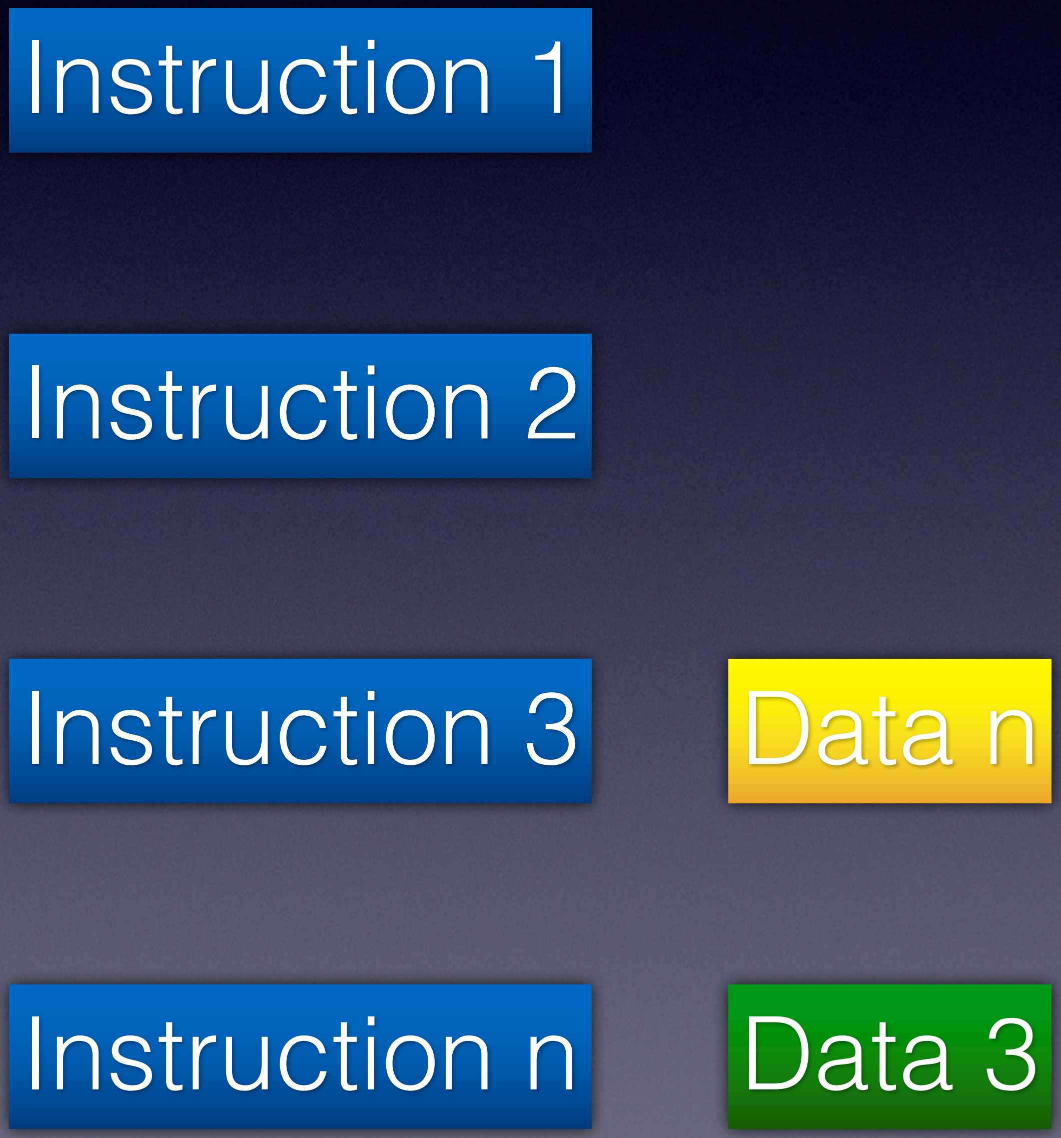
Pipeline program



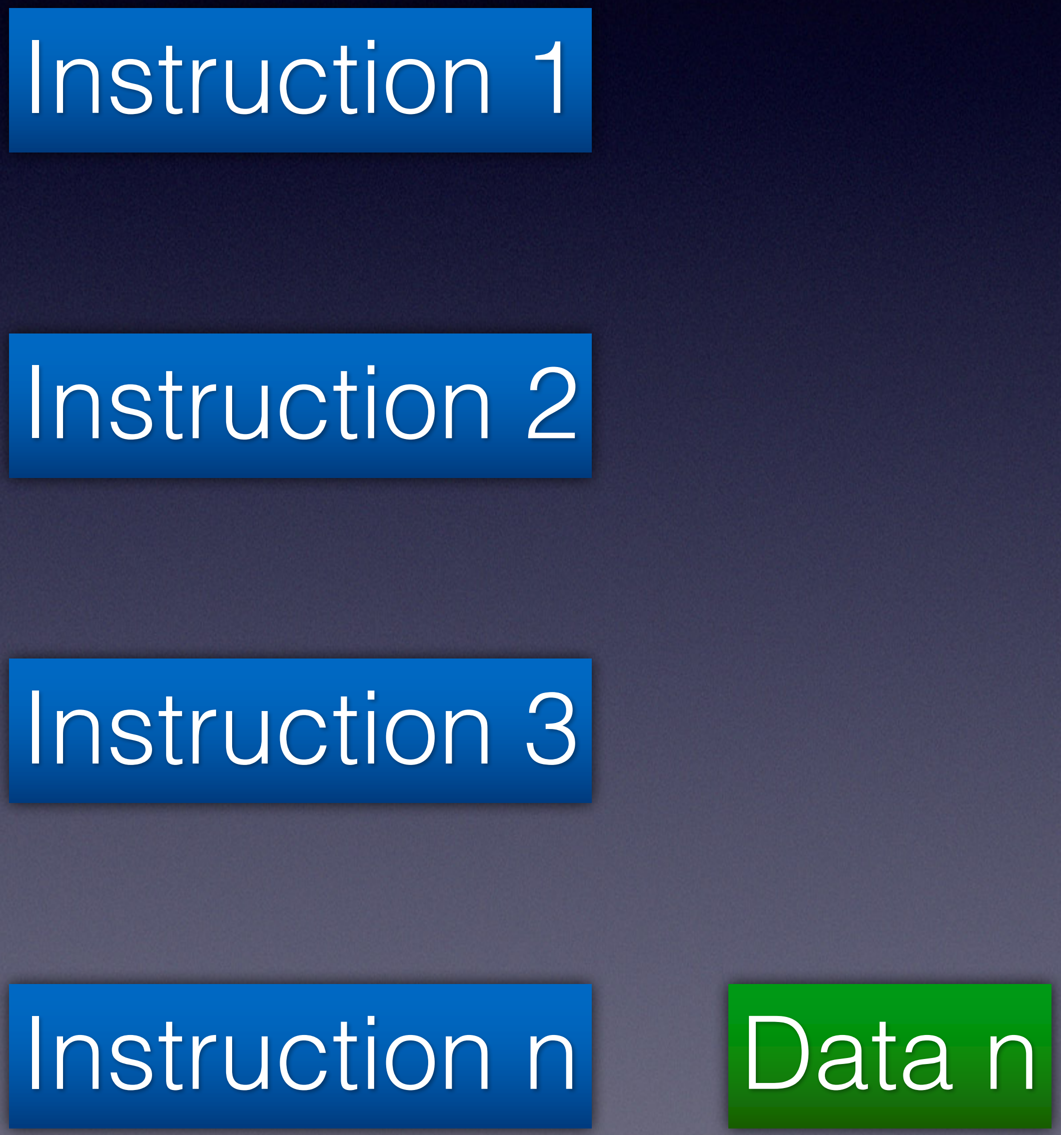
Pipeline program



Pipeline program



Pipeline program

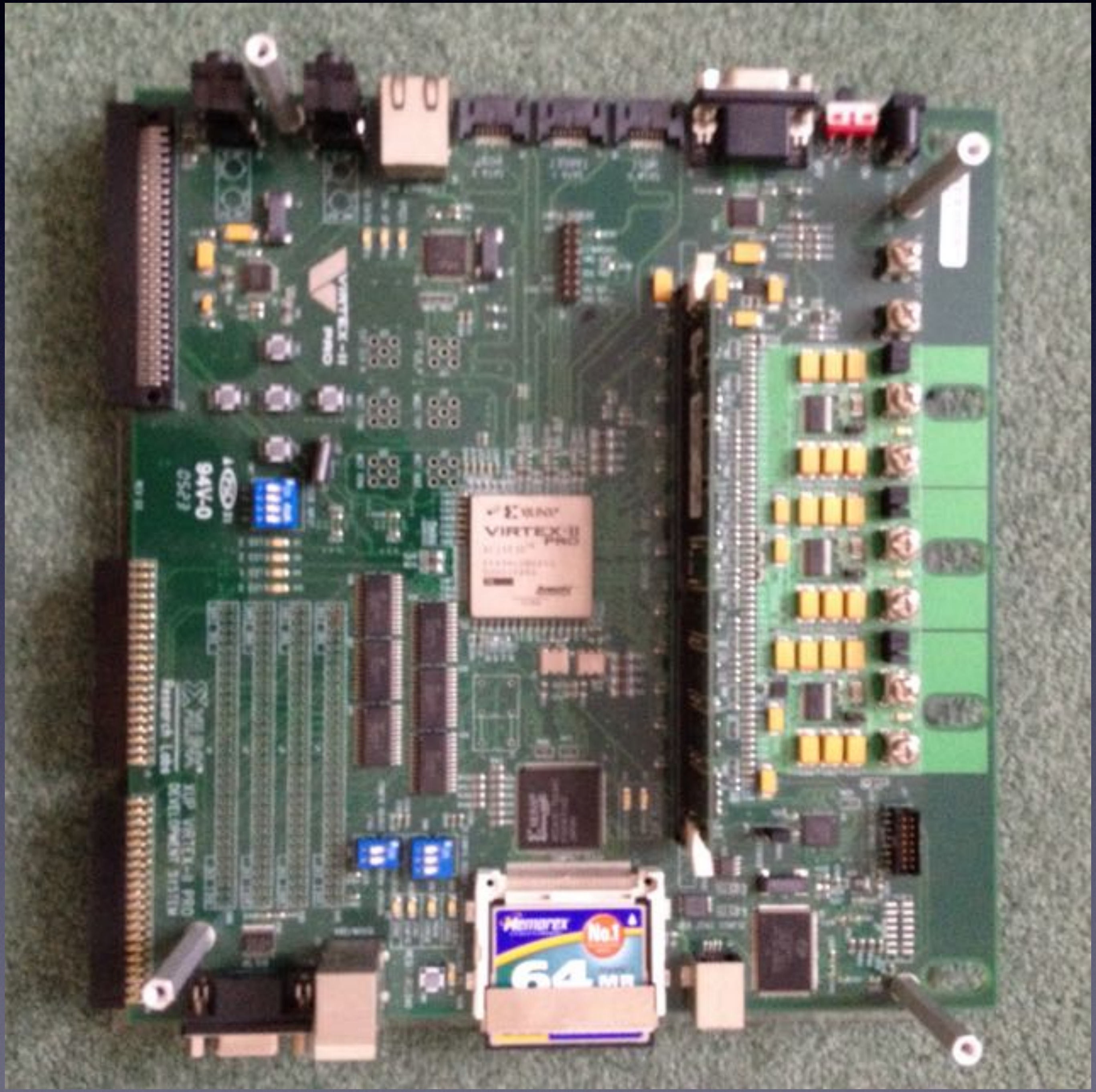


Hardware

Pipeline machine

- Throughput is independent of program length
- Data concurrency increases with program length
- The bigger the program, the faster it is - relative to a von-Neumann-core machine

FPGA prototype



Architectural prototype

- Token = 16 bit header + 80 bit float datum
- 64 k mills per chip
- 2 M mills per board
- 16 M mills per cabinet
- 20 kW per unweighted PWFLOP

Relative addressing

- Fixed size, relative address implements an address horizon in an arbitrarily large machine and maintains constant computational efficiency regardless of the size of the machine
- Small horizon keeps the token header small

Conditional indirection

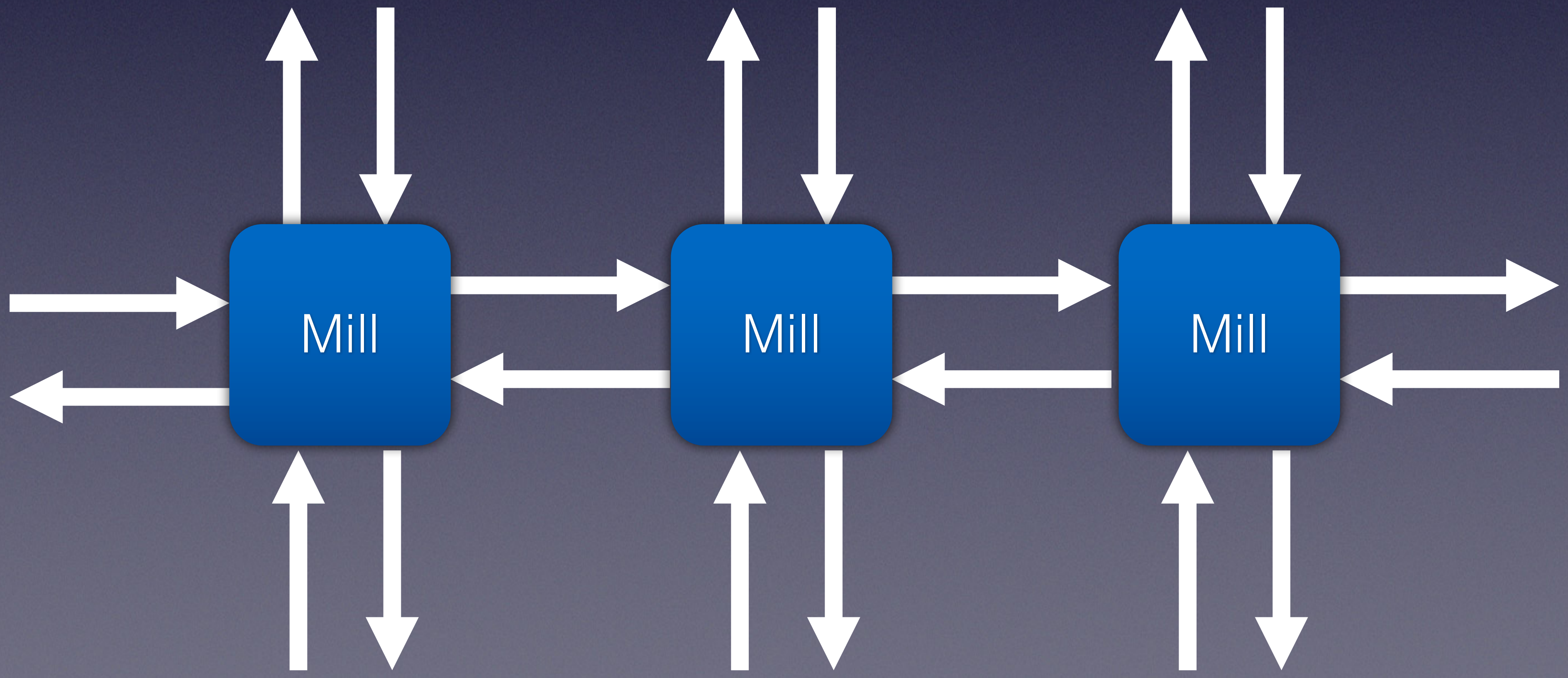
- Token header has one bit to signify delivery of the datum and one bit to signify redirection of the datum, with redirection address held in mill
- Thus two bits in the token header implement arbitrarily complex routing
- Keeps the token header small

Conditional execution

- Token header has one bit to signify execution of the instruction held in mill
- Keeps token header small

Architectural prototype

- Square array of mills
- Pipelined communication not just nearest neighbour

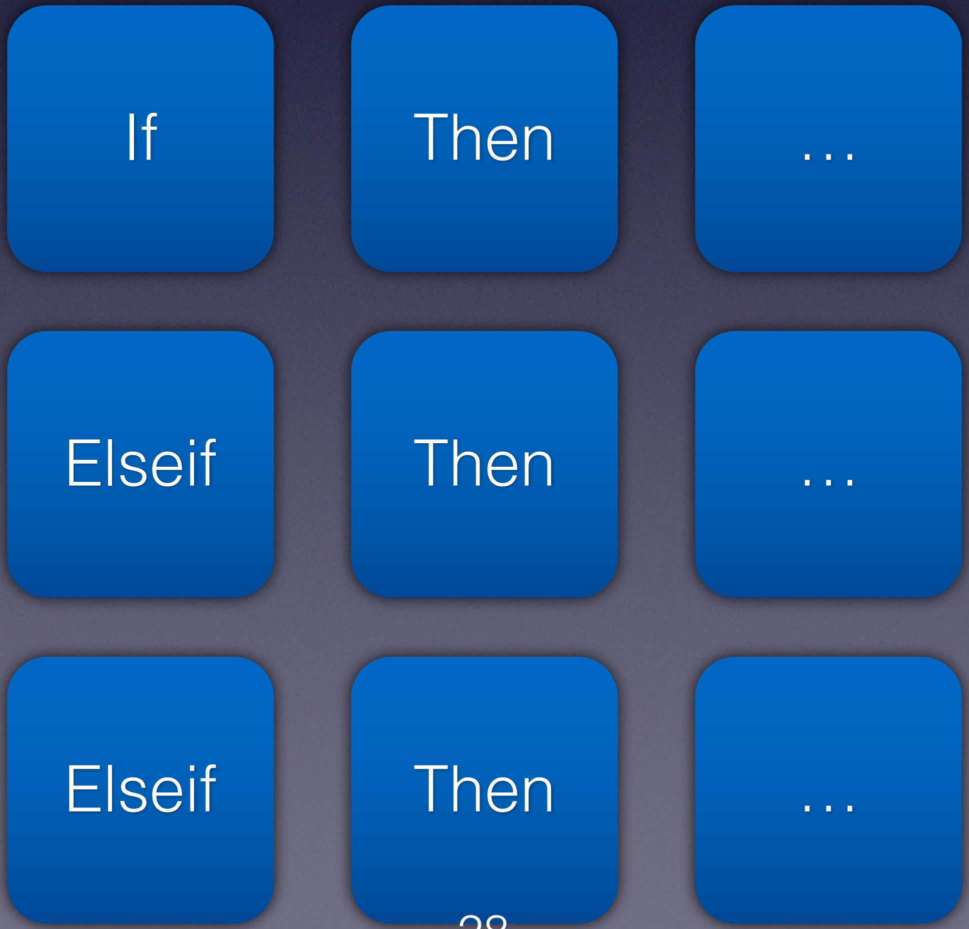


Pipelined communication

- Multiple mills emulate a von Neumann address space but with cycle time proportional to distance travelled
- Multiple mills emulate a systolic array but with bottleneck $O(n)$ on chip I/O
- Multiple mills implement a 2D pipeline with 1D I/O giving $O(1)$ bottleneck on chip I/O

2D pipeline

- Unbreakable if-then-elseif-…-elseif-else-endif pipeline



Known waiting time

- If the waiting time of an algorithm is known then its program can be laid out in an unbreakable 2D pipeline or slipstream
- Slipstream programs have a cadence of the slower of the times to effect the input or output of a record

Known waiting time

- Slipstream programs, with shared data, can execute concurrently
- Throughput = Cadence / Programs
- So an entire program can accumulate a result in less than one clock tick!

Molecular dynamics

- Many clock ticks to input one record to specify a molecule
- One clock tick to input one record to specify a molecule interaction
- Many clock ticks to output one record to update a molecule

Molecular dynamics

- 2 M mill board inputs 500 molecule specification records in many clock ticks
- Accumulates 500 molecule-molecule interactions per clock tick, over a stream of very many interaction records
- Outputs 500 molecule update records in many clock ticks
- Asymptotes to 500 program runs per clock tick

Unknown waiting time

- If code can be unrolled to one outer loop then data can be circulated through external memory and the body of the loop retains a known waiting time with all of the above advantages

- If loops cannot be unrolled then cadence is the longer of the I/O or loop-body times

Exception handling

- If machine is total then no logical system exceptions possible so no exception handling needed
- If waiting time is known then no programmer exception handling needed, just let a computational path halt and report after the waiting time
- Report physical faults on a schedule and roll back to preceding checkpoint
- Programmer reports unknown-waiting-time exceptions

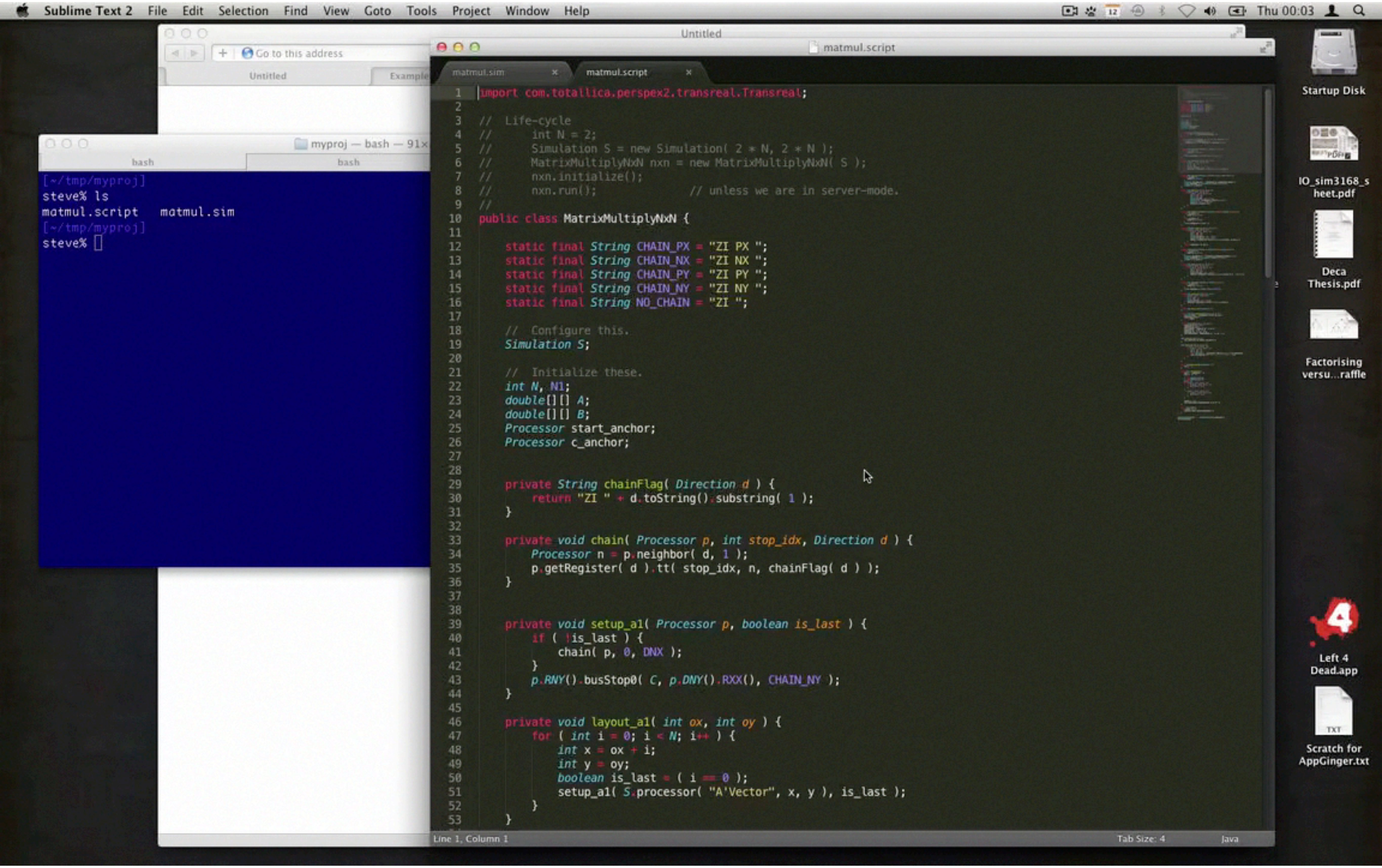
Software

Programming

- Architecture is Turing complete so any programming language can be used
- Map-Reduce programs, with data streams of known length, are guaranteed to be slipstreamable
- Von Neumann programs with counted loops, no recursion and no pointers are guaranteed to be slipstreamable

Demonstration

(click on next slide)



```
[~/tmp/myproj]
steve% ls
matmul.script   matmul.sim
[~/tmp/myproj]
steve% 
```

```
import com.totallica.perspex2.transreal.Transreal;

//  Life-cycle
//      int N = 2;
//      Simulation S = new Simulation( 2 * N, 2 * N );
//      MatrixMultiplyNxN nxn = new MatrixMultiplyNxN( S );
//      nxn.initialize();
//      nxn.run();              // unless we are in server-mode.
//
public class MatrixMultiplyNxN {

    static final String CHAIN_PX = "ZI PX ";
    static final String CHAIN_NX = "ZI NX ";
    static final String CHAIN_PY = "ZI PY ";
    static final String CHAIN_NY = "ZI NY ";
    static final String NO_CHAIN = "ZI ";

    //  Configure this.
    Simulation S;

    //  Initialize these.
    int N, N1;
    double[][] A;
    double[][] B;
    Processor start_anchor;
    Processor c_anchor;


    private String chainFlag( Direction d ) {
        return "ZI " + d.toString().substring( 1 );
    }

    private void chain( Processor p, int stop_idx, Direction d ) {
        Processor n = p.neighbor( d, 1 );
        p.getRegister( d ).tt( stop_idx, n, chainFlag( d ) );
    }


    private void setup_a1( Processor p, boolean is_last ) {
        if ( !is_last ) {
            chain( p, 0, DNX );
        }
        p.RNY().busStop0( C, p.DNY().RXX(), CHAIN_NY );
    }

    private void layout_a1( int ox, int oy ) {
        for ( int i = 0; i < N; i++ ) {
            int x = ox + i;
            int y = oy;
            boolean is_last = ( i == 0 );
            setup_a1( S.processor( "A'Vector", x, y ), is_last );
        }
    }
```

Summary

Relative addressing

- Reduces token size, making hardware more reliable and reducing power consumption
- Makes the machine scalable to any size, with constant efficiency

Conditional indirection

- Implements arbitrarily complex routing in just two header bits

2D addressing of mills

- Reduces token size, making hardware more reliable and reducing power consumption
- Branches laid out in space so no dynamic, branch-prediction failures
- Reduces time order of computations:

$$O(n^2) \rightarrow O(1)$$

Pipelined communication

- Emulates von Neumann addressing
- Emulates systolic addressing
- Delivers ideal data concurrency
- Can have long latency

Totality

- Reduces the number of program (model) errors
- Delivers unbreakable pipelines
- Removes all logical system errors
- Every syntactically correct program is semantically correct

General programming

- Von Neumann languages because Turing complete
- Systolic programming

Slipstream programming

- Von Neumann languages restricted to: counted loops, no pointers, no recursion
- Map-Reduce with known-length data-streams

Conclusion

- Power efficient because all tokens move a short distance per clock tick
- Scalable to any size with constant efficiency
- Safer code because totality removes many exceptions
- Enormous throughput of repeated computations
- Might deliver exascale on 20 MW power budget