Prospects for Functional Address Translation

Conor Hetland Georgios Tziantzioulis Brian Suchy

Kyle Hale Nikos Hardavellas Peter Dinda

Prescience Lab
Computer Science
Northwestern University

pdinda.org
presciencelab.org

Paper in a Nutshell

- Page tables implement address translation *functions*
- Can we implement these *functions* better on future (malleable) hardware?
  - Faster/more space efficient/easier to generate+update
- We studied four models for doing this
  - How well do they work for actual x64 page tables from HPC and other workloads?
- Results are mixed
  - Most promising technique: multiplexor tree

Outline

- Address translation functions
- Traditional paging and alternative models
- Our techniques and results
  - Perfect hashing for inverted page tables
  - Espresso-minimized PLAs
  - Task-specific functional language for FPGA synthesis
  - Multiplexor trees
- Conclusions and future work
  - CARAT

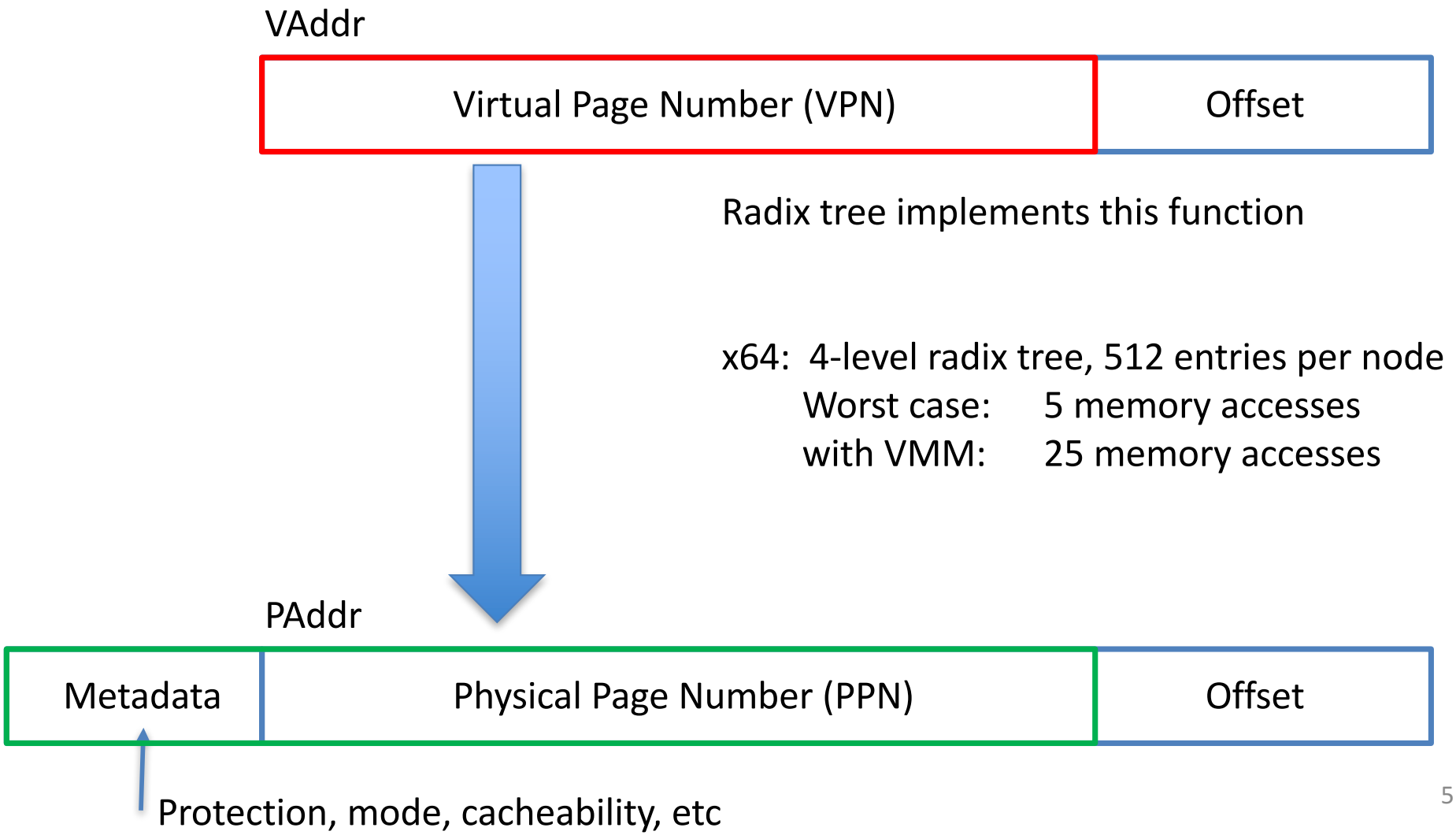
Motivation

- Address translation is a hot topic again
- Driven by… changing workloads
  - Database engines, cloud services, unikernels
- Driven by… TLB costs
  - TLB misses => lower performance
  - TLB reach may be insufficient
  - TLBs consume significant power/energy/area
  - Virtualization compounds these

- Our interest: parallel workloads and HPC
  - Part of a project to rethink the parallel hardware/software stack (interweaving.org)

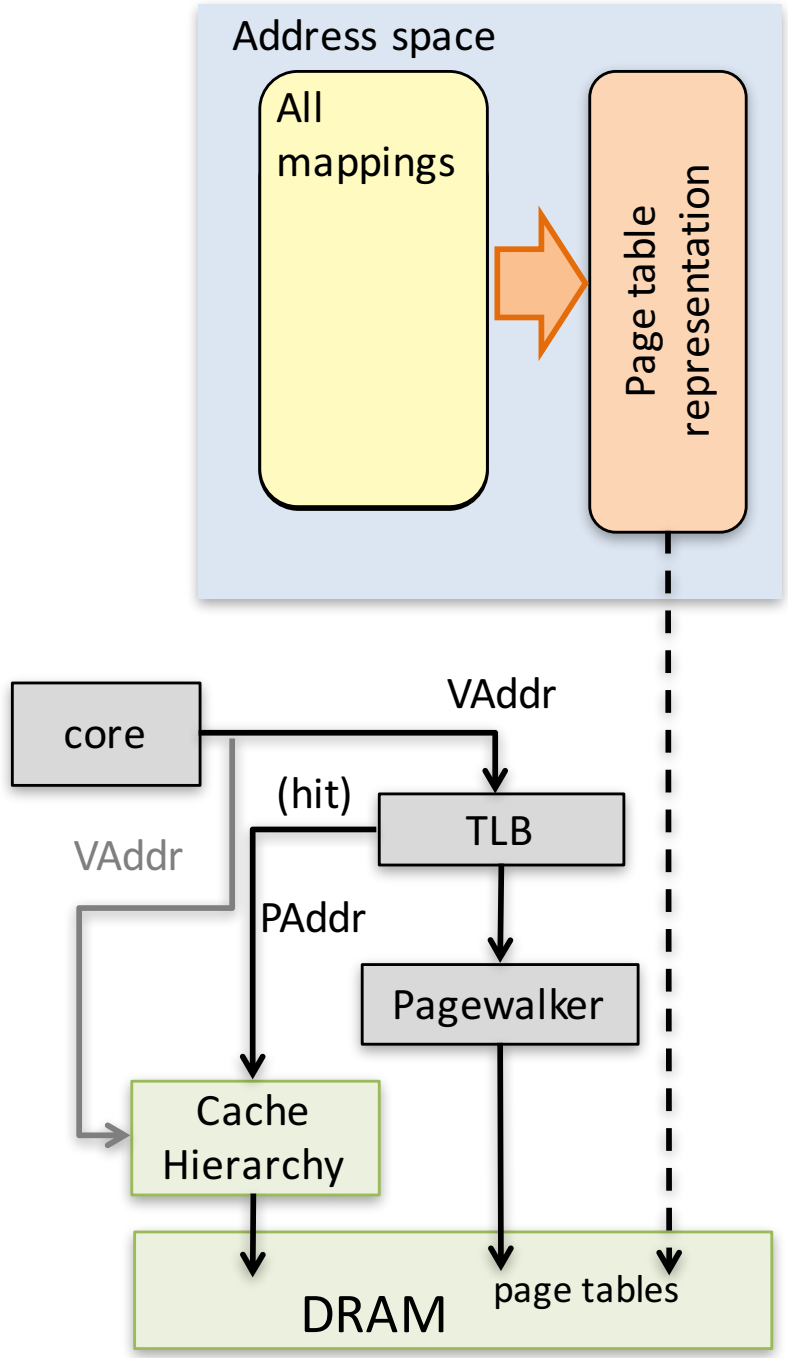
Address Translation *Functions*

Every Address (ifetch, data read/write/ control access / etc)



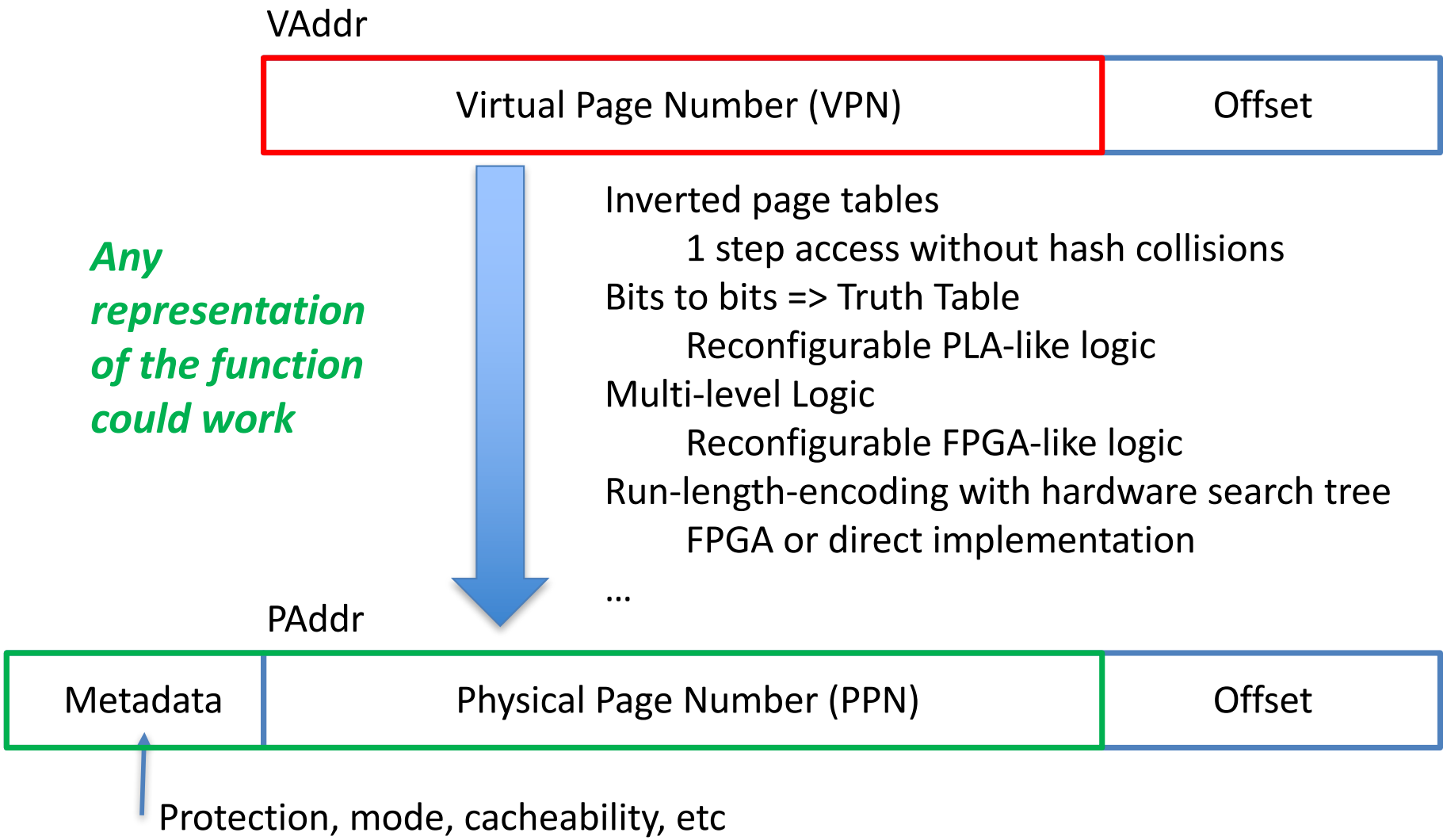
Traditional Paging (x64, others similar)

- TLB essential
  - Must avoid almost all actual page table references
- TLB design and cache design are tightly coupled, limiting cache design

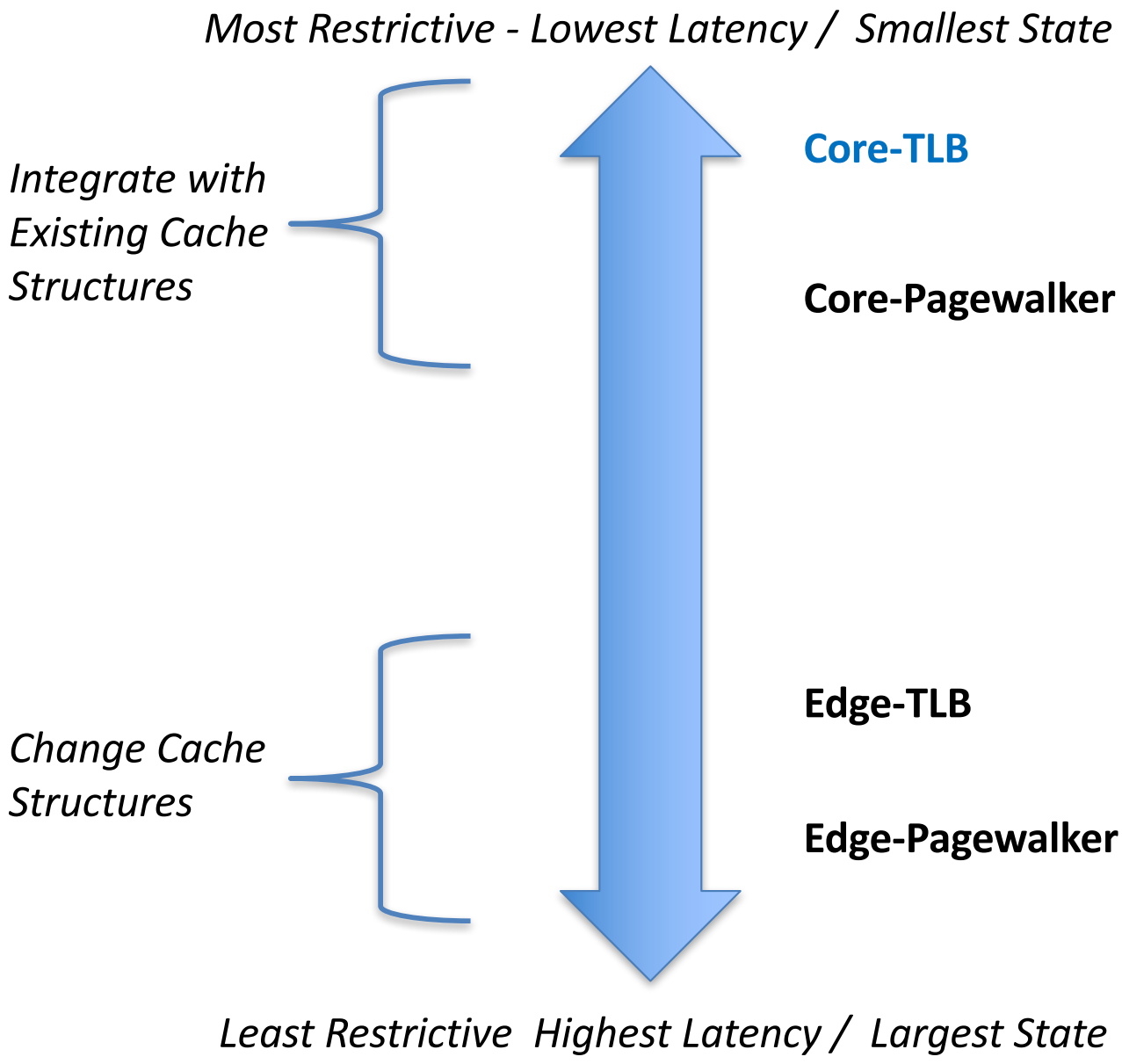


It Doesn't Need to be Page Tables

Every Address (ifetch, data read/write/ control access / etc)

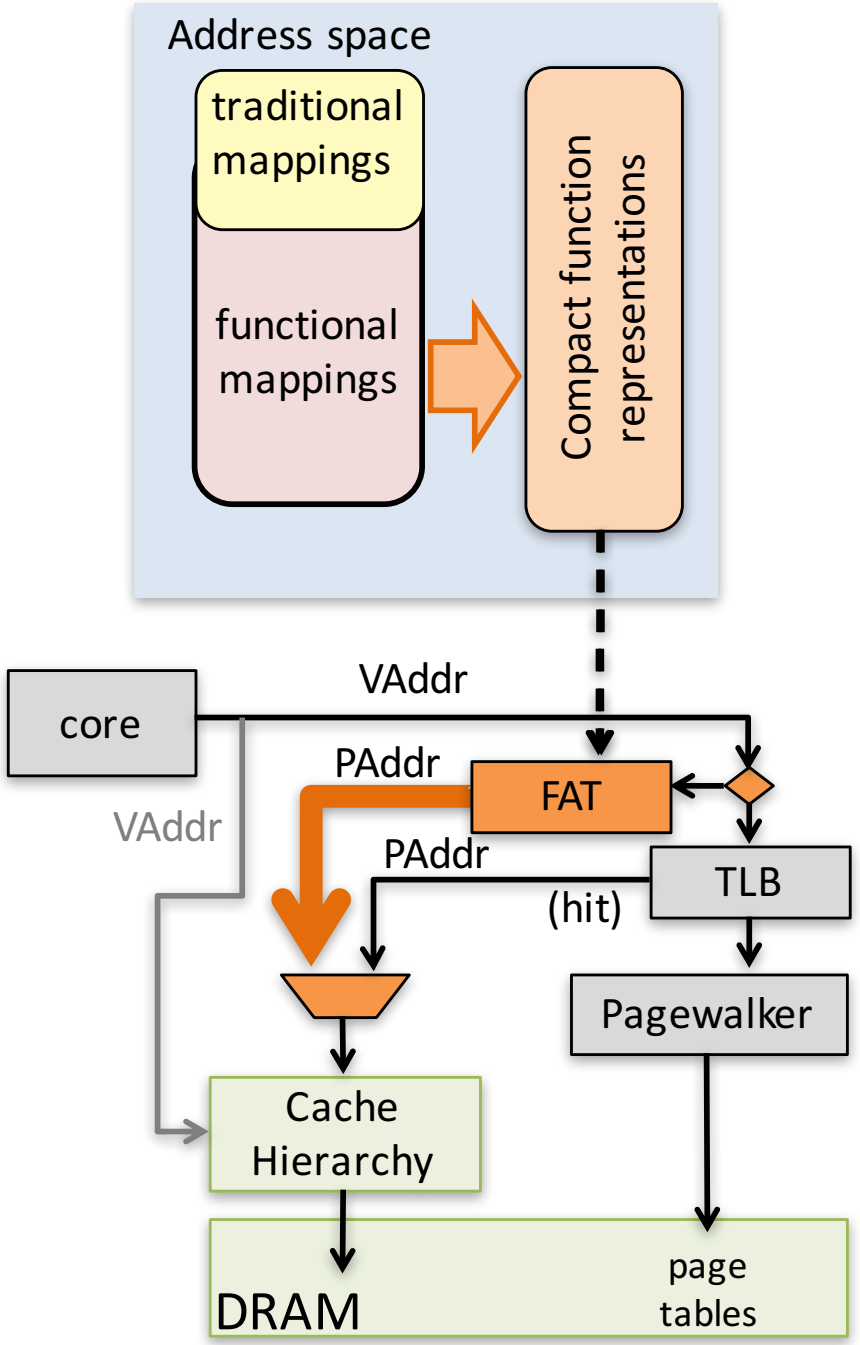


Locales for Functional Address Translation (FAT)



Core-TLB

- Most extreme
- Replace TLB!
  - Perhaps without misses!
- Must operate at least as fast as a TLB
  - Cannot have much state



Workloads

Our tool captures **Linux page table snapshots** (processes)

- General Purpose Servers
  - 2 machines, 19 days, ~1.2 million snapshots
- Mantevo
  - 13 snapshots, mid-run, of 13 benchmarks
- NAS
  - 40 snapshots, 4 each over 10 benchmarks
- PARSEC
  - 70 snapshots, 4-14 each over 7 benchmarks
- HPCCG on Legion
  - 221 snapshots, 1 second intervals
- Synthetic
  - 3 contrived snapshots

Evaluation Criteria

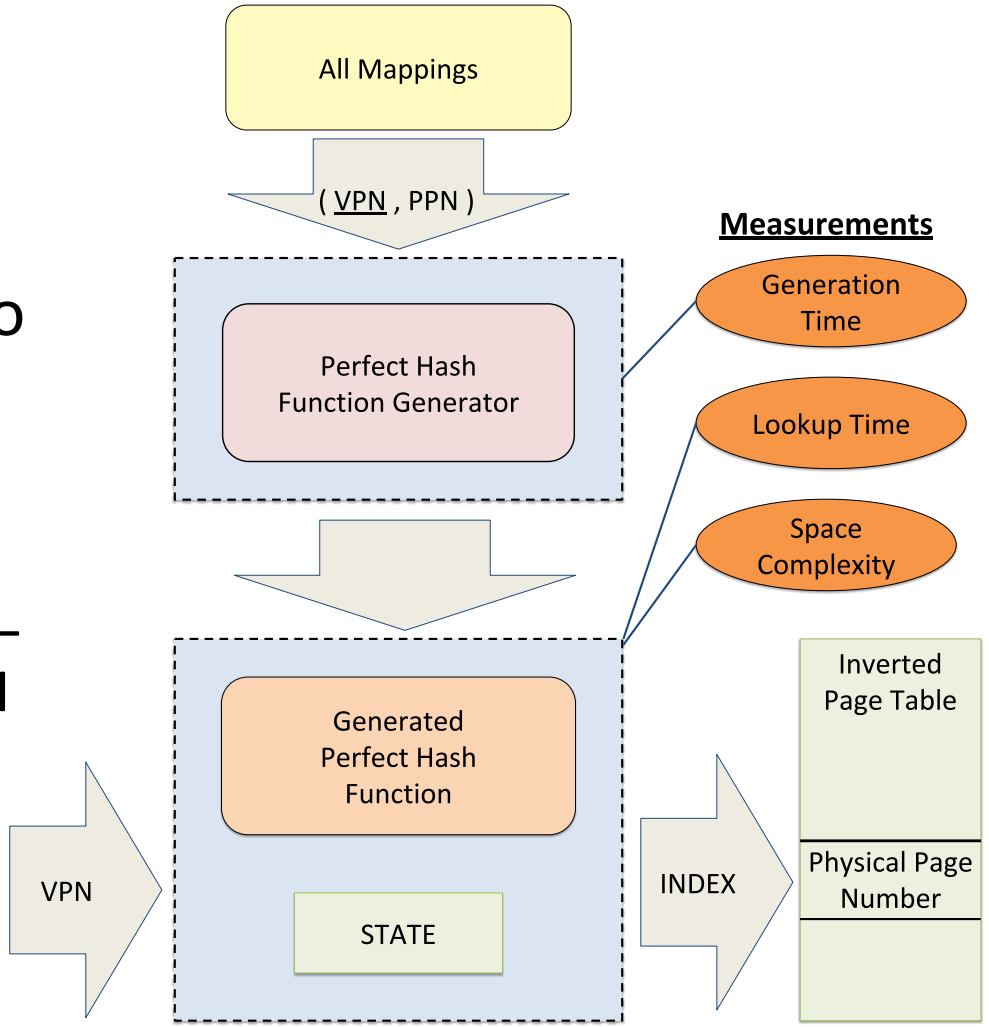
- Generation time
  - How quickly can we transform the mapping into the function?
    - By direct measurement of cost of software tool
- Space complexity
  - Estimate hardware resource cost
    - Ideally, by synthesis to FPGA via Verilog using Quartus, then counting logic blocks
- Lookup time
  - Estimate cost of lookup
    - Ideally by path length / cycle time of FPGA
- Caveat: update time not considered

Inverted Page Tables with Perfect Hashing

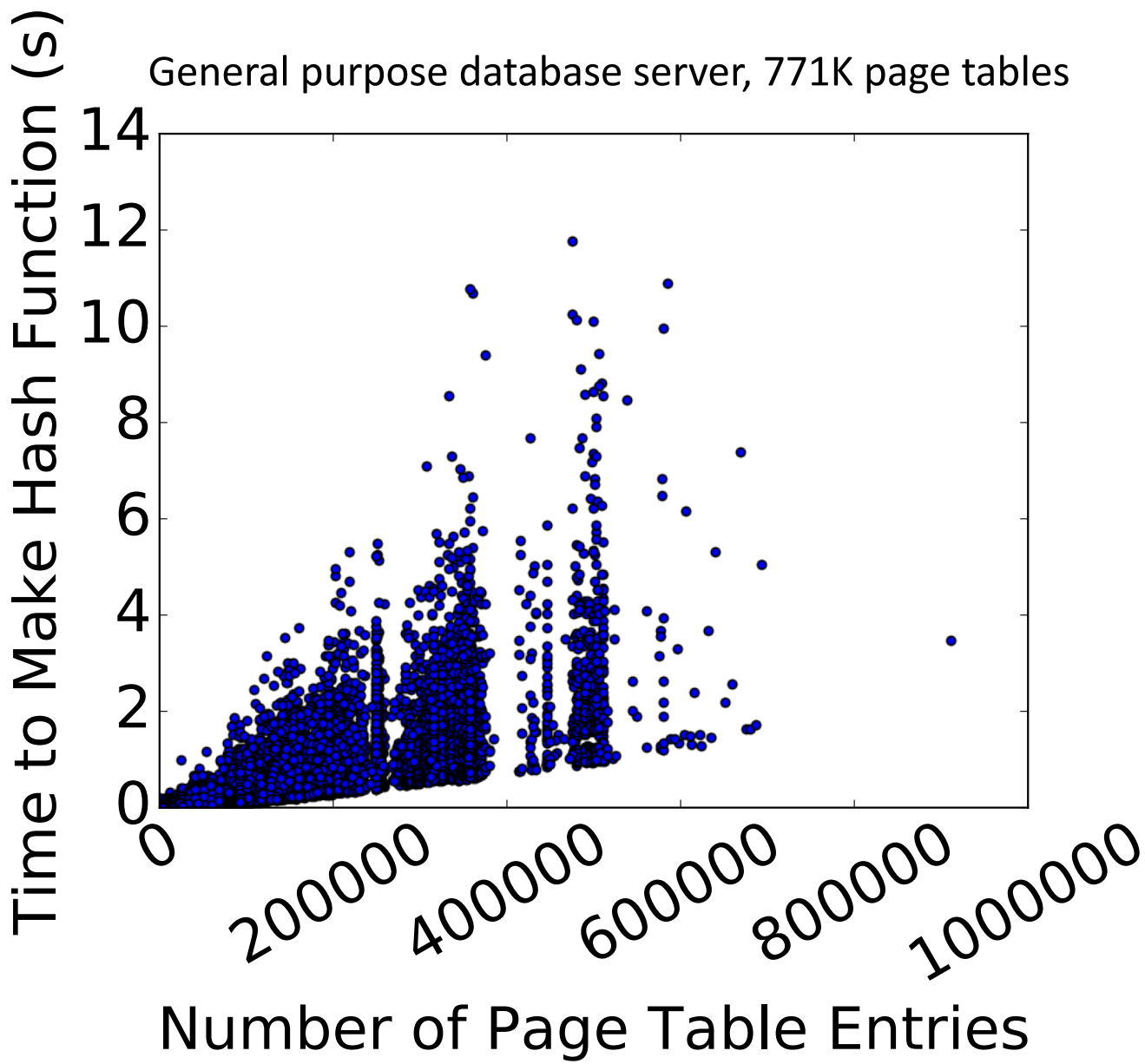
- IPT provides single-level page table entry lookup without hash collisions…
- …Perfect hashing makes hash functions that have no collisions

- Two generators considered
  - Minimum Perfect Hashing – CHM algorithm from CMPH
  - Perfect Hashing – GPERF

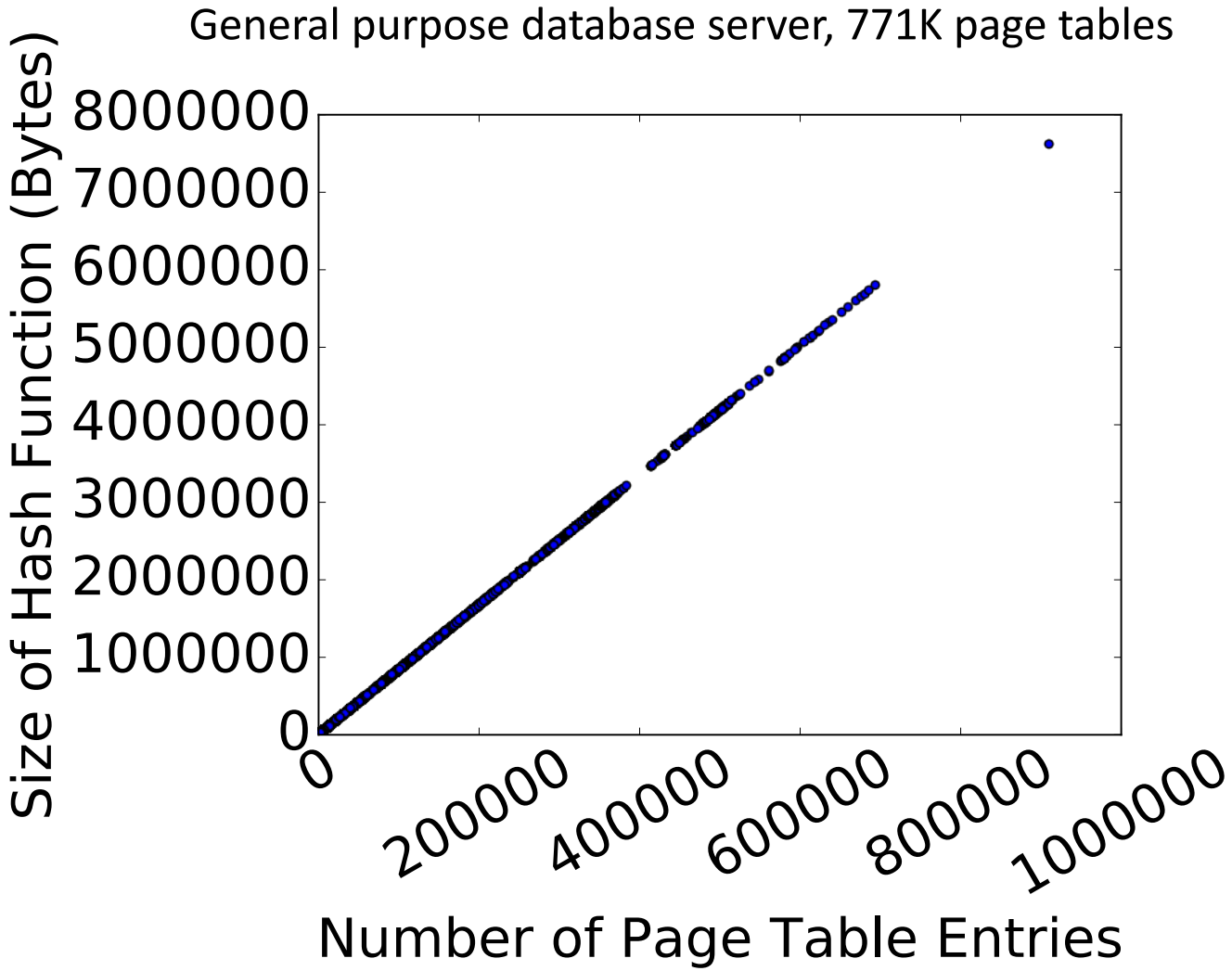
- CHM/CMPH is most interesting



Fast to Generate (CMPH/CHM)



Function Body Is Compact, But Needs Considerable State

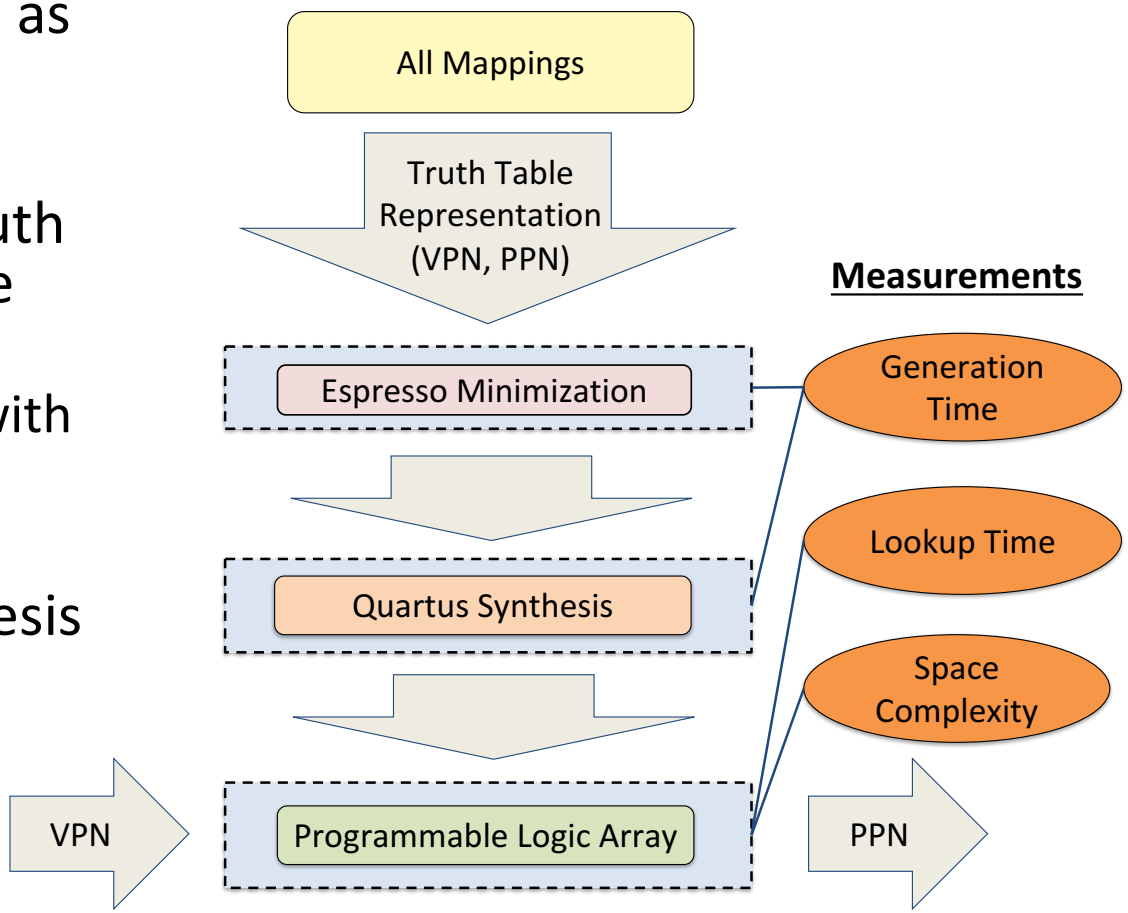


Lookups Are Tricky

- Hash function involves two memory references
  - With Inverse PT references, close to a forward PT…
- GPERF has contrasting results
  - **MUCH slower to generate** – no O(n) guarantee
  - Function body larger, but state much smaller
  - Lookup likely quite fast since state will not require memory accesses

Espresso-Minimized PLAs

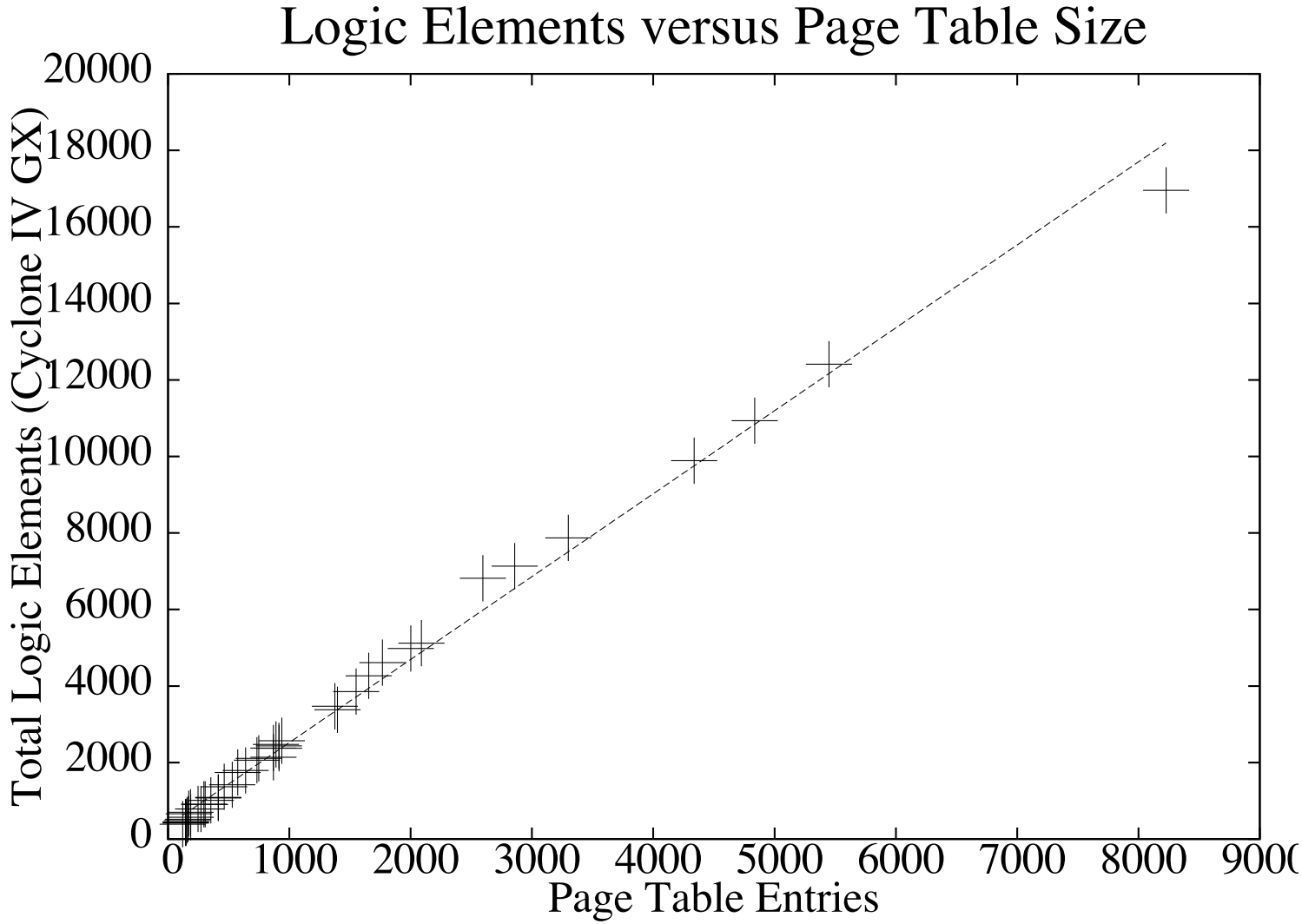
- Address translation function as a bit-by-bit truth table
- Classic Berkeley Espresso truth table minimizer for PLA-style reconfigurable logic
  - Imagine future processor with a PLA, not an FPGA
- For space complexity: synthesis by Quartus for FPGA
  - Limitation of evaluation
  - Real hardware would take Espresso output directly



Generation Costs Are High

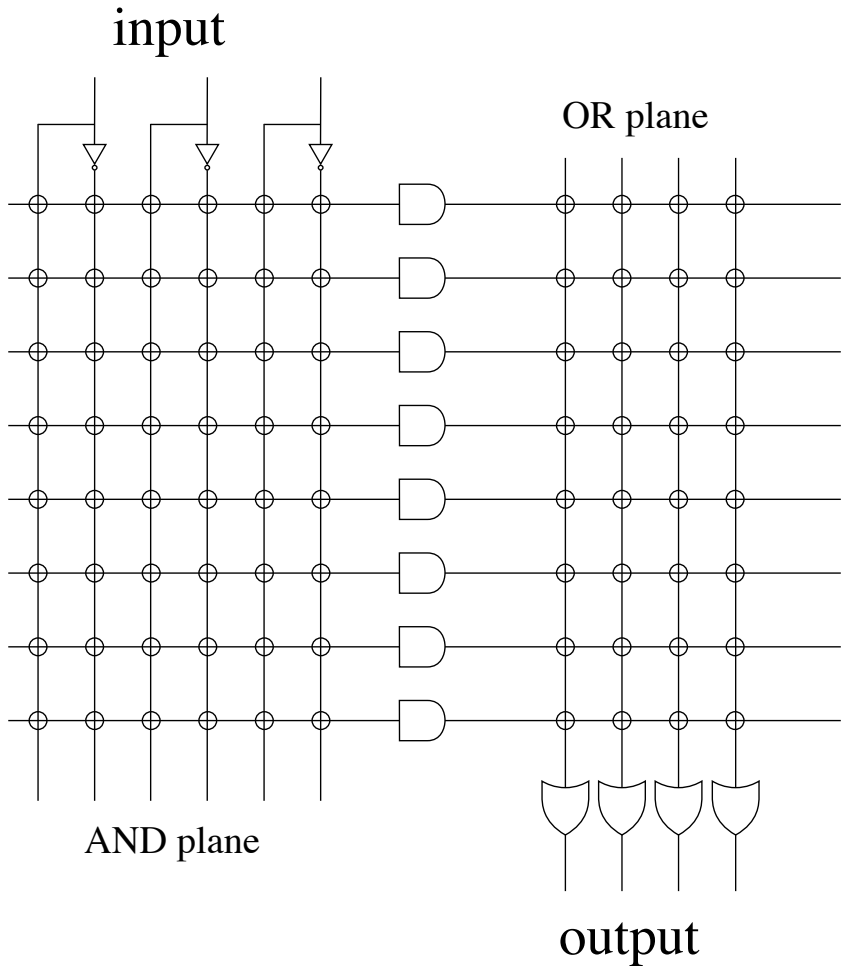
- 10s of minutes of CPU time for Espresso-minimization
  - Would always be needed
- Hours to synthesize into FPGA logic
  - Only needed for our evaluation approach
- Generation time limits the number of snapshots we can consider

Space Complexity Is Probably Linear in Address Space Size – Plus No Extra State

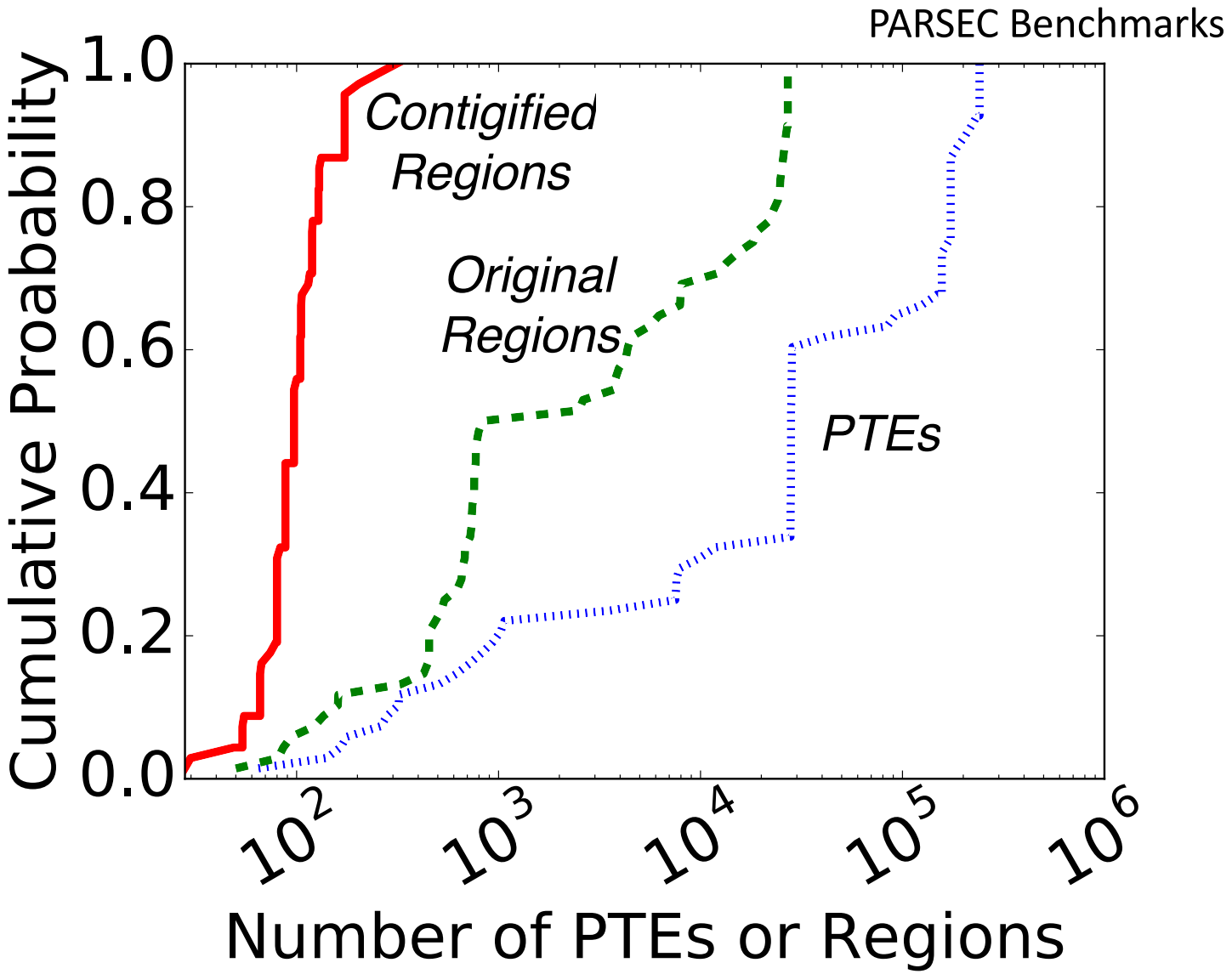


Lookup Time Likely Fast

- PLA is fundamentally two-level logic
- Provided sufficiently high fan-in/fan-out, lookup would be single-cycle
- Lots of caveats here…



Motivating Multiplexor Trees: Page Table Entries Respond Well to Run-length-encoding



Multiplexor Trees

- Think Linux memory map red-black tree…

(VPN, length) -> PPN

- … but realized as a hardware decision tree

A Two-Region Multiplexor Tree in Verilog

```
always @(vpn)
if ((vpn>=reg_vpn[0]) &&
    (vpn<(reg_vpn[0] + reg_num[0])))
     out = reg_ppn[0] + (vpn - reg_vpn[0]);
else
   if ((vpn>=reg_vpn[1]) &&
       (vpn<(reg_vpn[1] + reg_num[1])))
        out = reg_ppn[1] + (vpn - reg_vpn[1]);
   else
        out = 'hfffffffffffff;
```

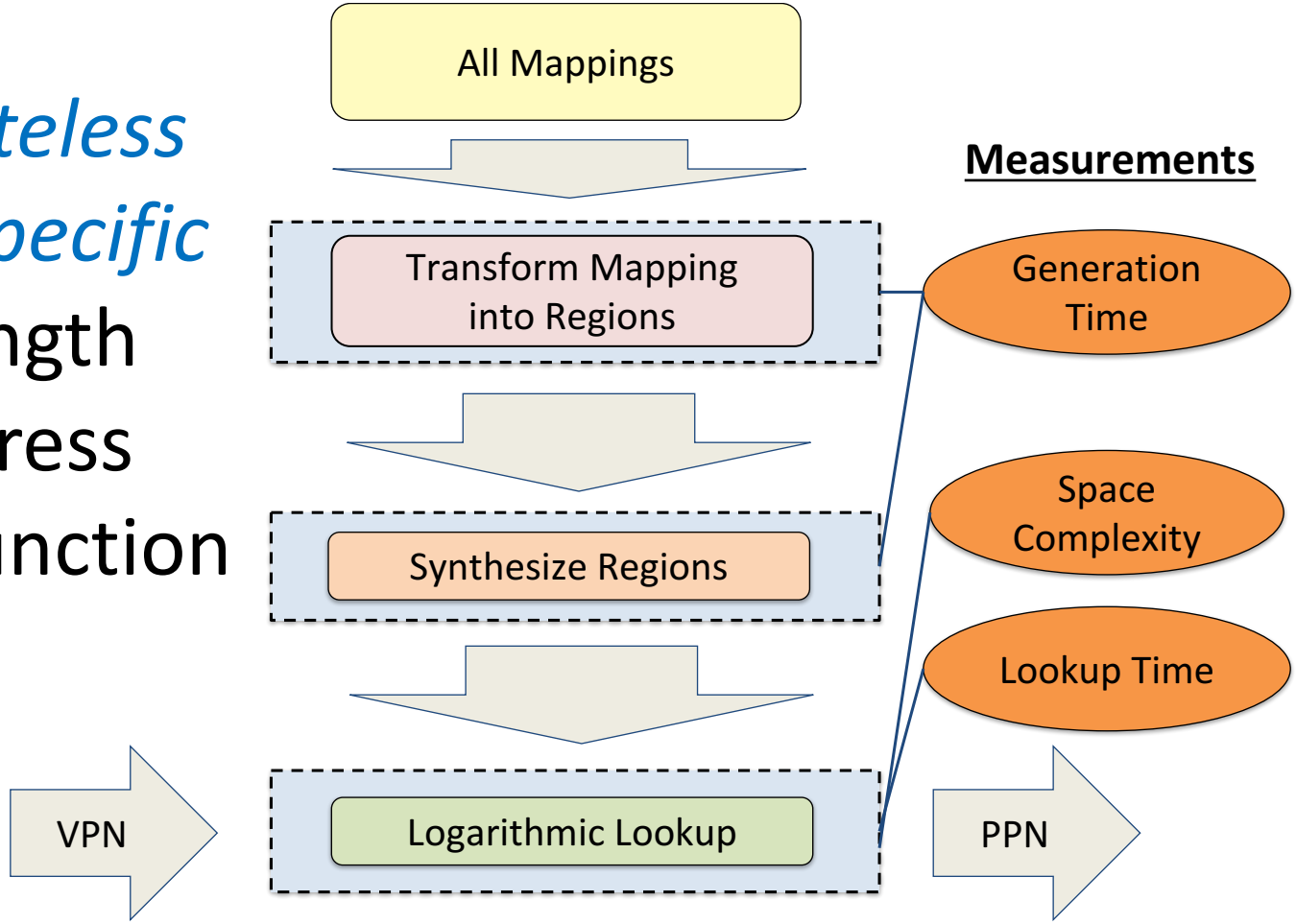
- Search process is this hardware tree

A 16-Region Multiplexor Tree in Verilog (trust me!)

```
always @(vpn)
        if ((vpn>=reg_vpn[7]) && (vpn<(reg_vpn[7]+reg_num[7])))
                out = reg_ppn[7] + (vpn-reg_vpn[7]);
        else if ((vpn<reg_vpn[7]))
                if ((vpn>=reg_vpn[3]) && (vpn<(reg_vpn[3]+reg_num[3])))
                        out = reg_ppn[3] + (vpn-reg_vpn[3]);
                else if ((vpn<reg_vpn[3]))
                        if ((vpn>=reg_vpn[1]) && (vpn<(reg_vpn[1]+reg_num[1])))
                                out = reg_ppn[1] + (vpn-reg_vpn[1]);
                        else if ((vpn<reg_vpn[1]))
                                if ((vpn>=reg_vpn[0]) && (vpn<(reg_vpn[0]+reg_num[0])))
                                        out = reg_ppn[0] + (vpn-reg_vpn[0]);
                                else
                                        out = 'hfffffffffffff;

                        else // vpn>=reg_vpn[1]+reg_num[1]
                                if ((vpn>=reg_vpn[2]) && (vpn<(reg_vpn[2]+reg_num[2])))
                                        out = reg_ppn[2] + (vpn-reg_vpn[2]);
                                else
                                        out = 'hfffffffffffff;


                else // vpn>=reg_vpn[3]+reg_num[3]
                        if ((vpn>=reg_vpn[4]) && (vpn<(reg_vpn[4]+reg_num[4])))
                                out = reg_ppn[4] + (vpn-reg_vpn[4]);
                        else if ((vpn<reg_vpn[4]))
                                if ((vpn>=reg_vpn[2]) && (vpn<(reg_vpn[2]+reg_num[2])))
                                        out = reg_ppn[2] + (vpn-reg_vpn[2]);
                                else
                                        if ((vpn>=reg_vpn[3]) && (vpn<(reg_vpn[3]+reg_num[3])))
                                                out = reg_ppn[3] + (vpn-reg_vpn[3]);
                                        else
                                                out = 'hfffffffffffff;


                        else // vpn>=reg_vpn[4]+reg_num[4]
                                if ((vpn>=reg_vpn[4]) && (vpn<(reg_vpn[4]+reg_num[4])))
                                        out = reg_ppn[4] + (vpn-reg_vpn[4]);
                                else if ((vpn<reg_vpn[4]))
                                        if ((vpn>=reg_vpn[3]) && (vpn<(reg_vpn[3]+reg_num[3])))
                                                out = reg_ppn[3] + (vpn-reg_vpn[3]);
                                        else
                                                out = 'hfffffffffffff;

                                else // vpn>=reg_vpn[4]+reg_num[4]
                                        if ((vpn>=reg_vpn[5]) && (vpn<(reg_vpn[5]+reg_num[5])))
                                                out = reg_ppn[5] + (vpn-reg_vpn[5]);
                                        else
                                                if ((vpn>=reg_vpn[6]) && (vpn<(reg_vpn[6]+reg_num[6])))
                                                        out = reg_ppn[6] + (vpn-reg_vpn[6]);
                                                else
                                                        out = 'hfffffffffffff;



else // vpn>=reg_vpn[7]+reg_num[7]
        if ((vpn>=reg_vpn[10]) && (vpn<(reg_vpn[10]+reg_num[10])))
                out = reg_ppn[10] + (vpn-reg_vpn[10]);
        else if ((vpn<reg_vpn[10]))
                if ((vpn>=reg_vpn[7]) && (vpn<(reg_vpn[7]+reg_num[7])))
                        out = reg_ppn[7] + (vpn-reg_vpn[7]);
                else if ((vpn<reg_vpn[7]))
                        if ((vpn>=reg_vpn[6]) && (vpn<(reg_vpn[6]+reg_num[6])))
                                out = reg_ppn[6] + (vpn-reg_vpn[6]);
                        else
                                out = 'hfffffffffffff;

                else // vpn>=reg_vpn[7]+reg_num[7]
                        if ((vpn>=reg_vpn[8]) && (vpn<(reg_vpn[8]+reg_num[8])))
                                out = reg_ppn[8] + (vpn-reg_vpn[8]);
                        else
                                if ((vpn>=reg_vpn[9]) && (vpn<(reg_vpn[9]+reg_num[9])))
                                        out = reg_ppn[9] + (vpn-reg_vpn[9]);
                                else
                                        out = 'hfffffffffffff;


        else // vpn>=reg_vpn[10]+reg_num[10]
                if ((vpn>=reg_vpn[12]) && (vpn<(reg_vpn[12]+reg_num[12])))
                        out = reg_ppn[12] + (vpn-reg_vpn[12]);
                else if ((vpn<reg_vpn[12]))
                        if ((vpn>=reg_vpn[10]) && (vpn<(reg_vpn[10]+reg_num[10])))
                                out = reg_ppn[10] + (vpn-reg_vpn[10]);
                        else if ((vpn<reg_vpn[10]))
                                if ((vpn>=reg_vpn[9]) && (vpn<(reg_vpn[9]+reg_num[9])))
                                        out = reg_ppn[9] + (vpn-reg_vpn[9]);
                                else
                                        out = 'hfffffffffffff;

                        else // vpn>=reg_vpn[10]+reg_num[10]
                                if ((vpn>=reg_vpn[11]) && (vpn<(reg_vpn[11]+reg_num[11])))
                                        out = reg_ppn[11] + (vpn-reg_vpn[11]);
                                else
                                        out = 'hfffffffffffff;


                else // vpn>=reg_vpn[12]+reg_num[12]
                        if ((vpn>=reg_vpn[13]) && (vpn<(reg_vpn[13]+reg_num[13])))
                                out = reg_ppn[13] + (vpn-reg_vpn[13]);
                        else if ((vpn<reg_vpn[13]))
                                if ((vpn>=reg_vpn[11]) && (vpn<(reg_vpn[11]+reg_num[11])))
                                        out = reg_ppn[11] + (vpn-reg_vpn[11]);
                                else
                                        if ((vpn>=reg_vpn[12]) && (vpn<(reg_vpn[12]+reg_num[12])))
                                                out = reg_ppn[12] + (vpn-reg_vpn[12]);
                                        else
                                                out = 'hfffffffffffff;


                        else // vpn>=reg_vpn[13]+reg_num[13]
                                if ((vpn>=reg_vpn[13]) && (vpn<(reg_vpn[13]+reg_num[13])))
                                        out = reg_ppn[13] + (vpn-reg_vpn[13]);
                                else if ((vpn<reg_vpn[13]))
                                        if ((vpn>=reg_vpn[12]) && (vpn<(reg_vpn[12]+reg_num[12])))
                                                out = reg_ppn[12] + (vpn-reg_vpn[12]);
                                        else
                                                out = 'hfffffffffffff;

                                else // vpn>=reg_vpn[13]+reg_num[13]
                                        if ((vpn>=reg_vpn[14]) && (vpn<(reg_vpn[14]+reg_num[14])))
                                                out = reg_ppn[14] + (vpn-reg_vpn[14]);
                                        else
                                                if ((vpn>=reg_vpn[15]) && (vpn<(reg_vpn[15]+reg_num[15])))
                                                        out = reg_ppn[15] + (vpn-reg_vpn[15]);
                                                else
                                                        out = 'hfffffffffffff;
```

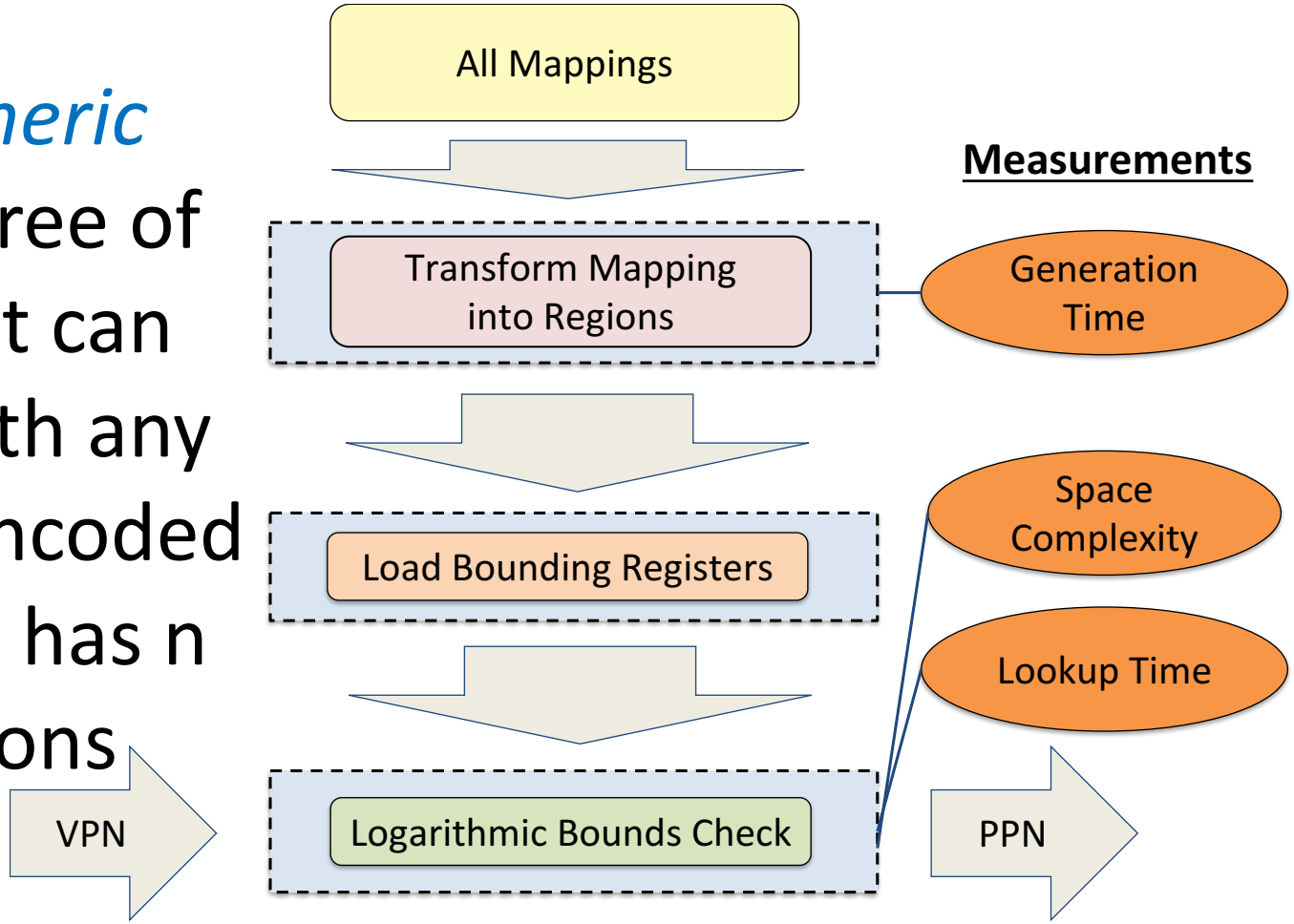
Bespoke Multiplexor Trees

- Generate *stateless* search tree *specific* to the run-length encoded address translation function



Registered Multiplexor Trees

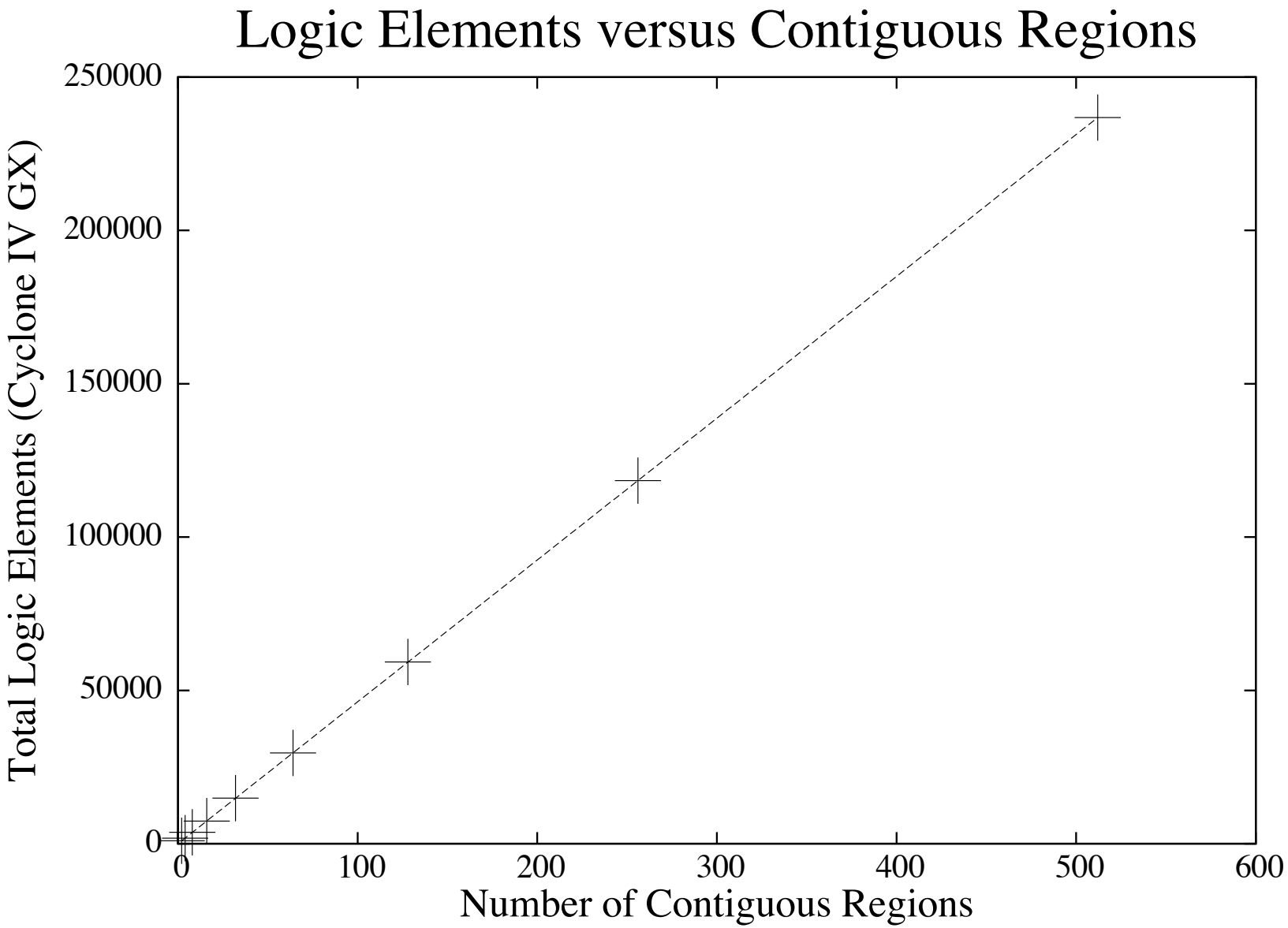
- Generate *generic* multiplexor tree of n regions that can be *loaded* with any run-length-encoded function that has n or fewer regions



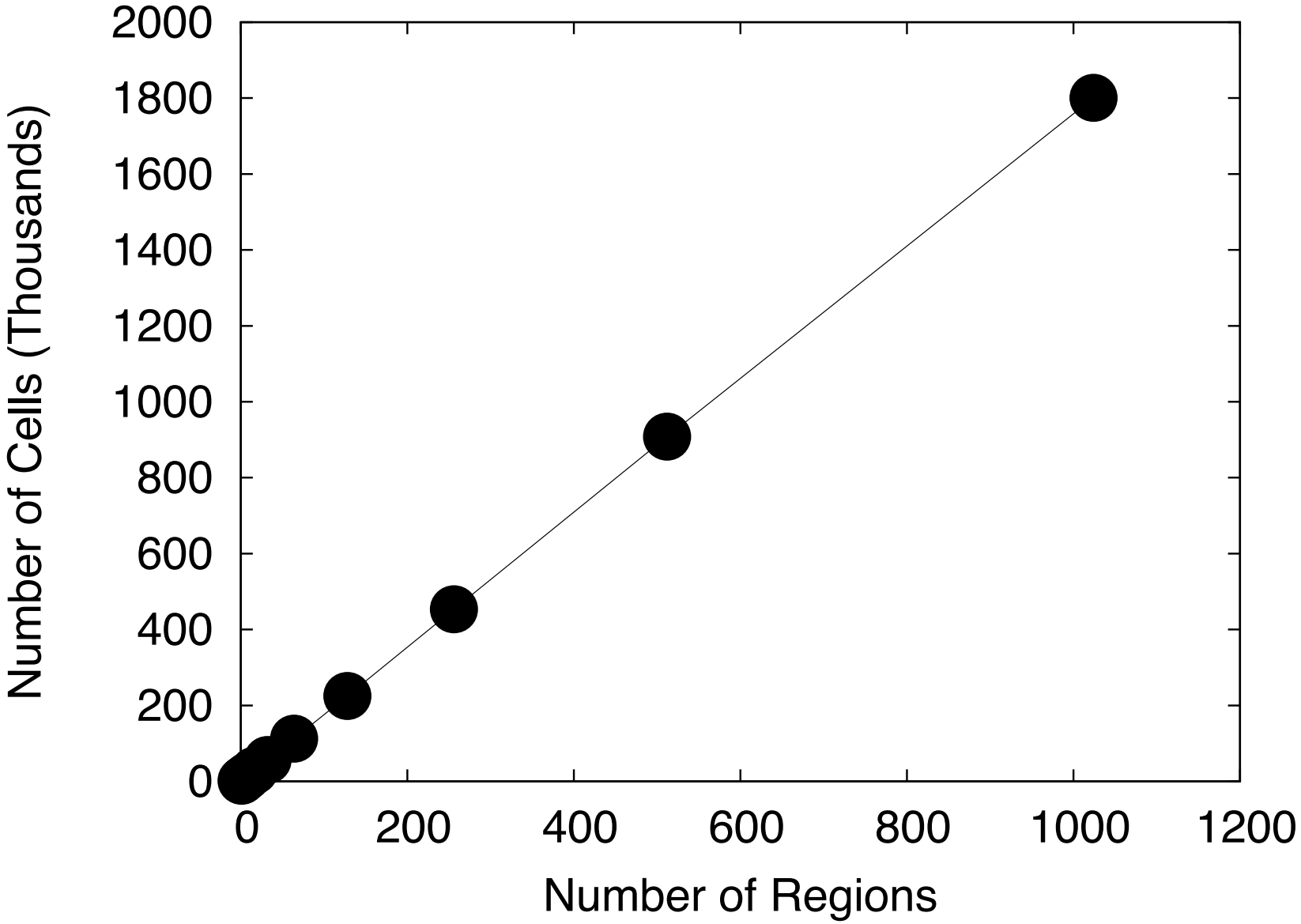
Generation Costs

- Generally high (hours)
  - Milliseconds to generate, hours to synthesize
- Matters for bespoke
- Does not matter for registered
  - We can reuse the synthesized hardware just by loading its registers with other values

Space Complexity is Reasonable (Bespoke)



Space Complexity is Reasonable (Registered)



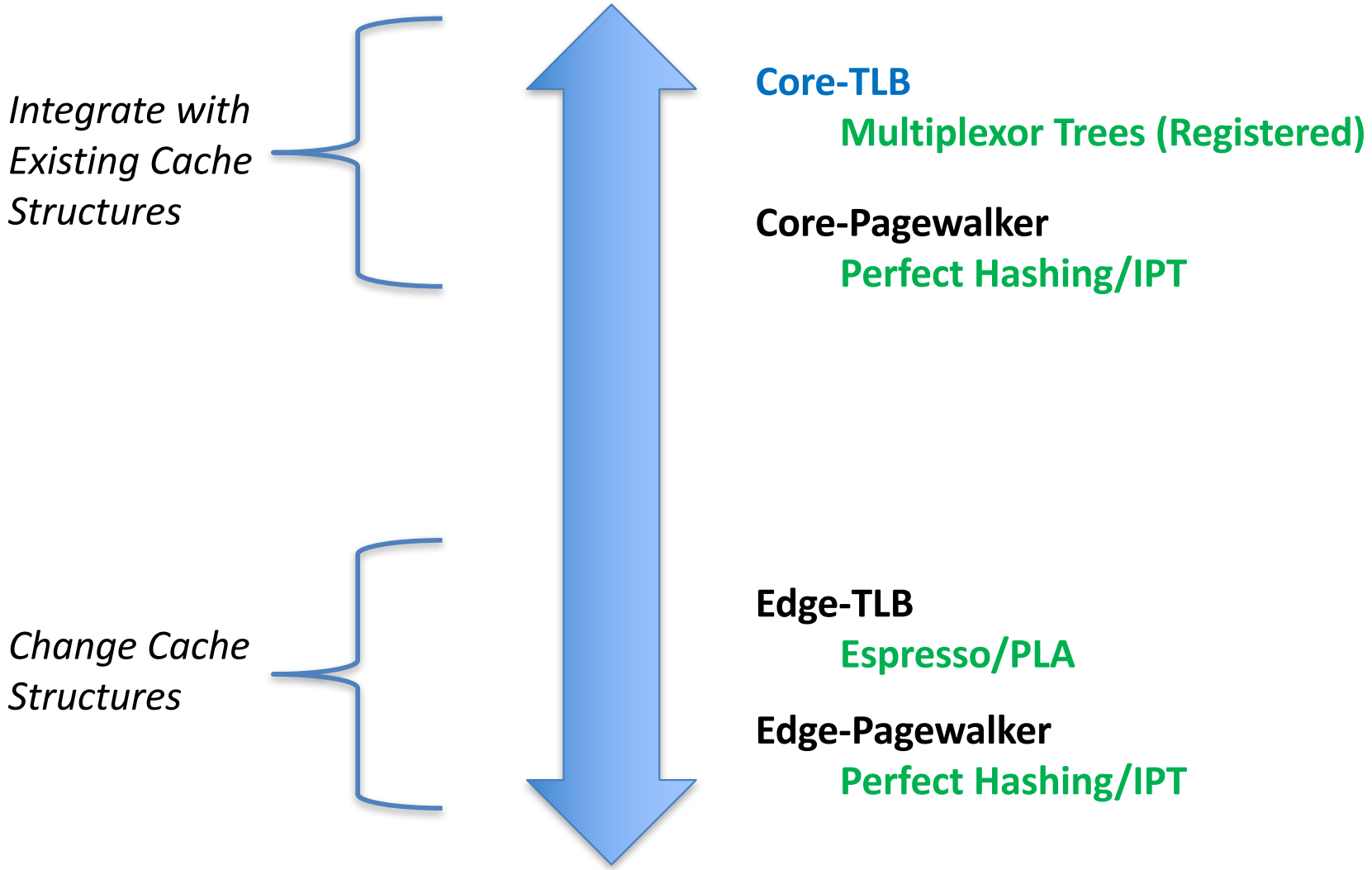
Lookup Cost

- Log depth circuit (n=number of regions)
- Single cycle for what we synthesized
  - But this is for a slow FPGA
  - And depends on scale
- Bespoke versus registered does not matter

- Kernel can potentially control these costs through memory allocation model
  - Enhance virtual->physical contiguity -> ”contigify”

Speculation About Best Fits

Most Restrictive - Lowest Latency / Smallest State



Least Restrictive Highest Latency / Largest State

Related Work

- DIY Address Translation (Alam et al, ISCA 17)
- Hash, Don’t Cache (Yaniv et al, SIGMETRICS 16)
- Translation Caching/SpecTLB (Barr et al, ISCA 10)
- Segments / RLE tables (Basu et al, ISCA 13)
- Nested Paging (Bhargava et al, ASPLOS 08)
- Bhattacharjee et al (numerous, present)
- Numerous ideas from the last time address translation was a hot topic (mid ‘90s)

Current Direction: CARAT

- Compiler- and Runtime-based Address Translation

- ***No*** address translation – all ***physical*** addressing
  - Optionally – no page abstraction
- Compiler and runtime jointly enforce protection and enable data mobility
- Paper forthcoming!

Paper in a Nutshell

- Page tables implement address translation *functions*
- Can we implement these *functions* better on future (malleable) hardware?
  - Faster/more space efficient/easier to generate+update
- We studied four models for doing this
  - How well do they work for actual x64 page tables from HPC and other workloads?
- Results are mixed
  - Most promising technique: multiplexor tree

For More Information

- Peter Dinda
  - pdinda@northwestern.edu
  - http://pdinda.org
- Interweaving Project
  - http://interweaving.org
- Prescience Lab
  - http://presciencelab.org
- Acknowledgements
  - NSF, DOE, Intel