ILLINOIS

CSL | Coordinated Science Lab

COLLEGE OF ENGINEERING

Subho S. Banerjee, Zbigniew T. Kalbarczyk and Ravishankar K. Iyer

Computer Science, Electrical and Computer Engineering

UIUC

AcMC^2: Accelerated Markov Chain Monte Carlo for Probabilistic Models

ASPLOS 2019

Probabilistic Models: Core of Many AI Apps

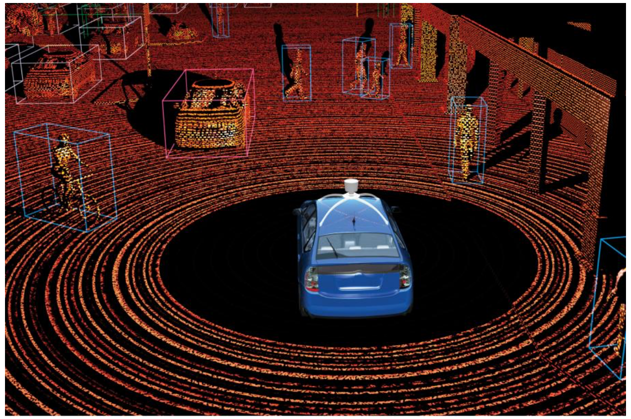
- Probabilistic modeling: integrates domain knowledge, quantifies uncertainties

Probabilistic Models: Core of Many AI Apps

- Probabilistic modeling: integrates domain knowledge, quantifies uncertainties
- Probabilistic programs: Encode probability models

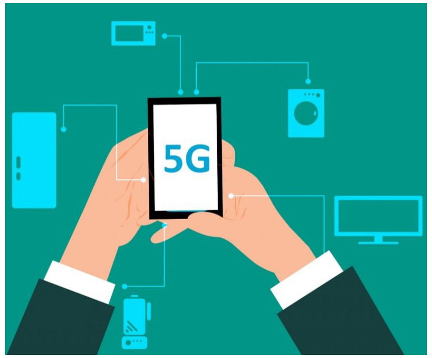
Probabilistic Models: Core of Many AI Apps

- Probabilistic modeling: integrates domain knowledge, quantifies uncertainties
- Probabilistic programs: Encode probability models



Sensor Fusion in Self Driving Vehicles

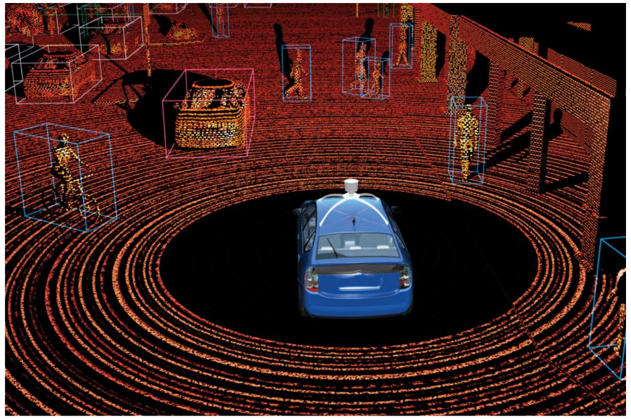
Skill Matching in Online Gaming (TrueSkill 1&2 from Microsoft)



4G or 5G Communication Devices (Turbo/LDPC Codes)

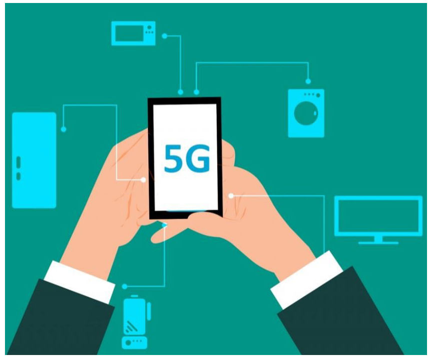
Probabilistic Models: Core of Many AI Apps

- Probabilistic modeling: integrates domain knowledge, quantifies uncertainties
- Probabilistic programs: Encode probability models



Sensor Fusion in Self Driving Vehicles

Skill Matching in Online Gaming (TrueSkill 1&2 from Microsoft)

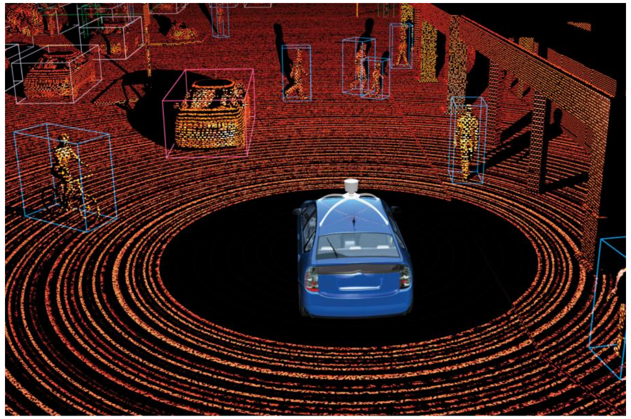


4G or 5G Communication Devices (Turbo/LDPC Codes)

- Inference: General solutions based on Markov Chain Monte Carlo

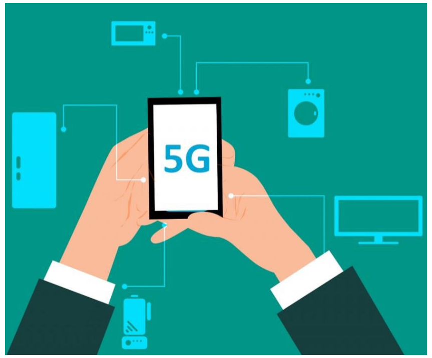
Probabilistic Models: Core of Many AI Apps

- Probabilistic modeling: integrates domain knowledge, quantifies uncertainties
- Probabilistic programs: Encode probability models



Sensor Fusion in Self Driving Vehicles

Skill Matching in Online Gaming (TrueSkill 1&2 from Microsoft)



4G or 5G Communication Devices (Turbo/LDPC Codes)

- Inference: General solutions based on Markov Chain Monte Carlo
- Extremely compute intensive & real time constraints

Our Approach

Automatically generate efficient accelerator from high level description

1. Abstraction: Domain specific languages

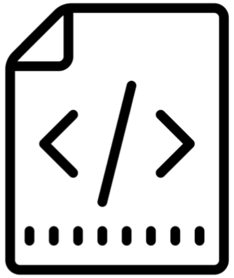
2. Mapping abstractions to an architecture

Our Approach

Automatically generate efficient accelerator from high level description

1. Abstraction: Domain specific languages
2. **Mapping abstractions to an architecture**

Probabilistic Programming Languages



BLOG
Stan
Church
Tensorflow Prob.
Pyro (Pytorch)

Our Approach

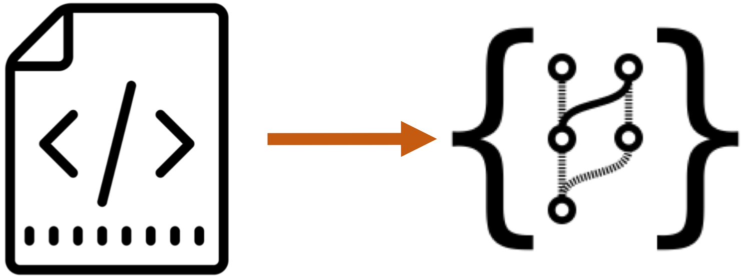
Automatically generate efficient accelerator from high level description

1. Abstraction: Domain specific languages

2. Mapping abstractions to an architecture

Probabilistic Programming Languages

Probabilistic Graphical Model based IR



BLOG
Stan
Church
Tensorflow Prob.
Pyro (Pytorch)

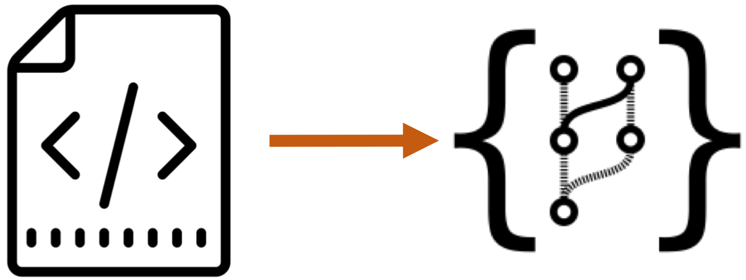
Our Approach

Automatically generate efficient accelerator from high level description

1. Abstraction: Domain specific languages
2. **Mapping abstractions to an architecture**

Probabilistic Programming Languages

Probabilistic Graphical Model based IR



BLOG
Stan
Church
Tensorflow Prob.
Pyro (Pytorch)

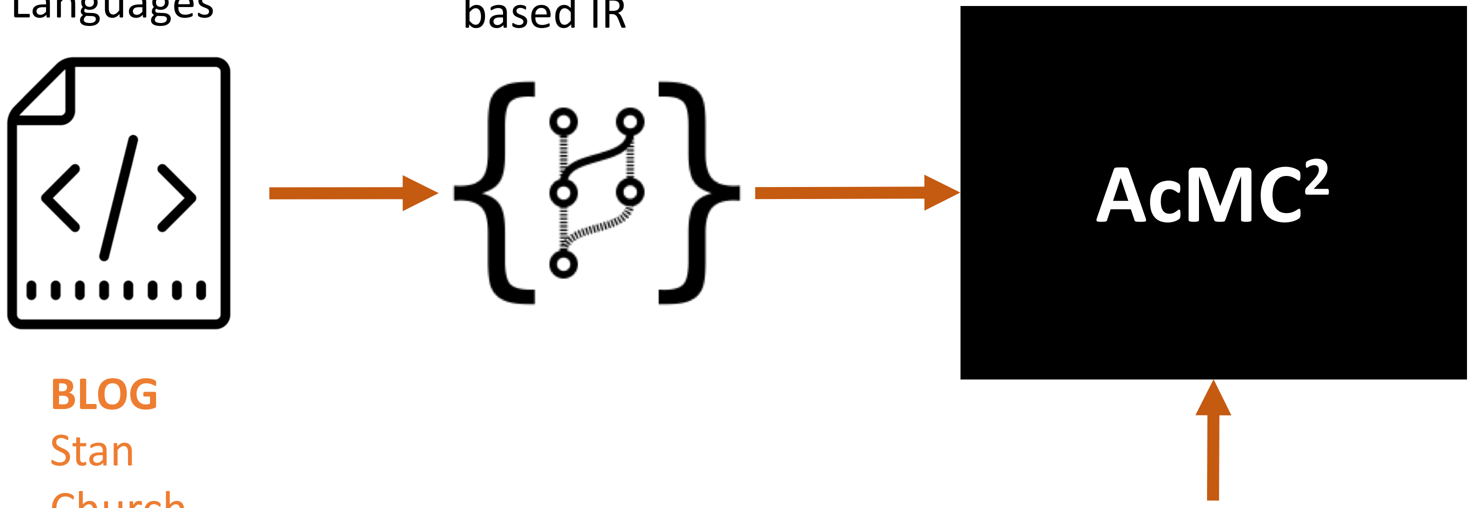
Our Approach

Automatically generate efficient accelerator from high level description

1. Abstraction: Domain specific languages
2. **Mapping abstractions to an architecture**

Probabilistic Programming Languages

Probabilistic Graphical Model based IR



BLOG
Stan
Church
Tensorflow Prob.
Pyro (Pytorch)

Inference Procedure
Markov Chain Monte Carlo

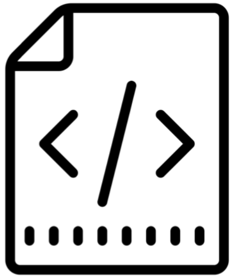
Our Approach

Automatically generate efficient accelerator from high level description

1. Abstraction: Domain specific languages

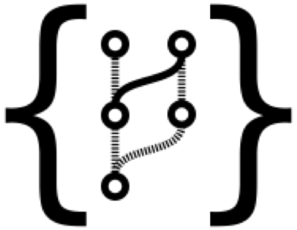
2. Mapping abstractions to an architecture

Probabilistic Programming Languages



BLOG
Stan
Church
Tensorflow Prob.
Pyro (Pytorch)

Probabilistic Graphical Model based IR



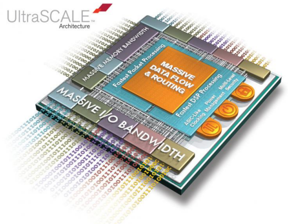
AcMC²

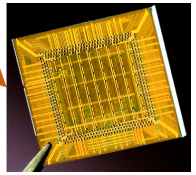
Inference Procedure
Markov Chain Monte Carlo

Traditional synthesis flow

CHISEL
VIVADO
SYNOPSYS

FPGAs





ASICs

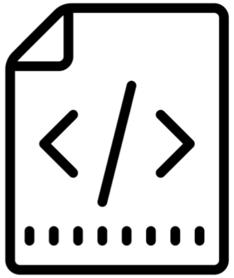
Our Approach

Automatically generate efficient acce
1. Abstraction: Domai
2. Mapping abstractio

Contributions:

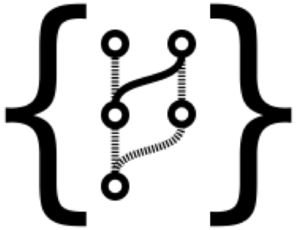
1. Identify accelerable kernels
2. Opportunities for parallelism
3. Knobs for trading off accuracy and performance

Probabilistic Programming Languages



BLOG
Stan
Church
Tensorflow Prob.
Pyro (Pytorch)

Probabilistic Graphical Model based IR

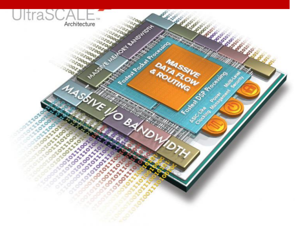


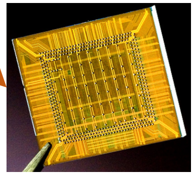
AcMC²

Inference Procedure
Markov Chain Monte Carlo

synthesis flow

CHISEL
VIVADO
SYNOPSYS





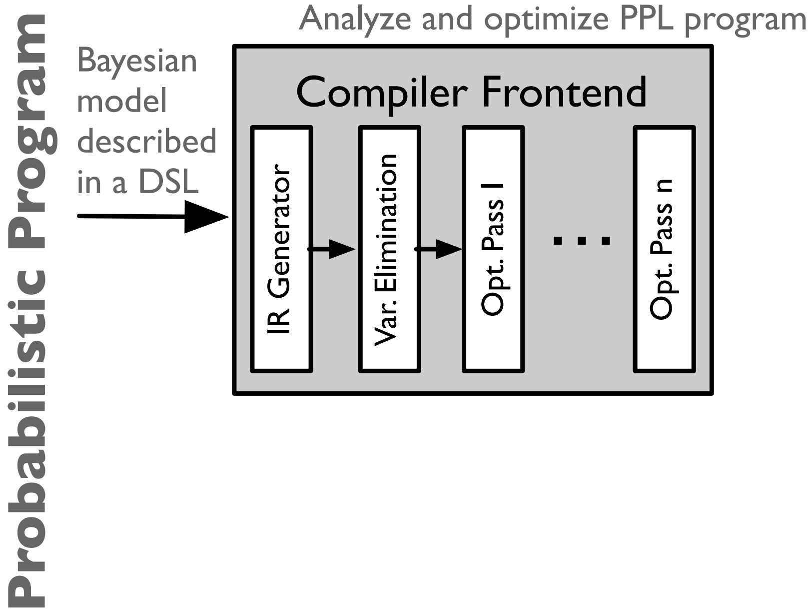
ASICs

AcMC² Deep Dive

- Generic architecture for MCMC accelerators
  - Efficient high dimensional random number generators (samplers)
- Specializes architecture of accelerator given model, inference method

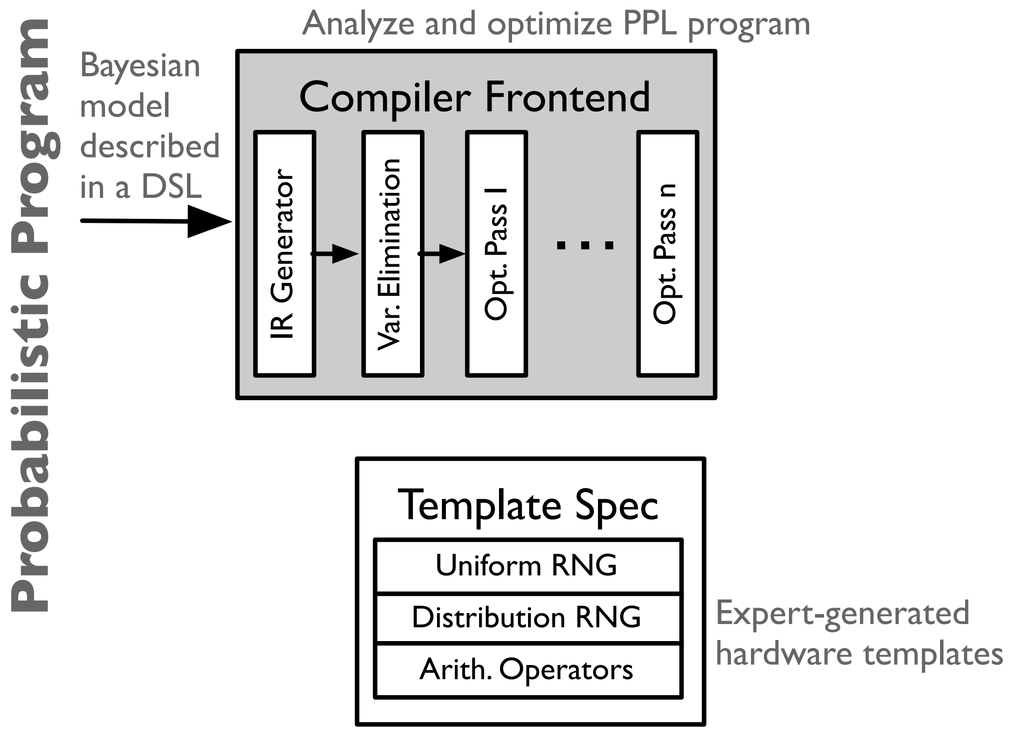
AcMC² Deep Dive

- Generic architecture for MCMC accelerators
  - Efficient high dimensional random number generators (samplers)
- Specializes architecture of accelerator given model, inference method



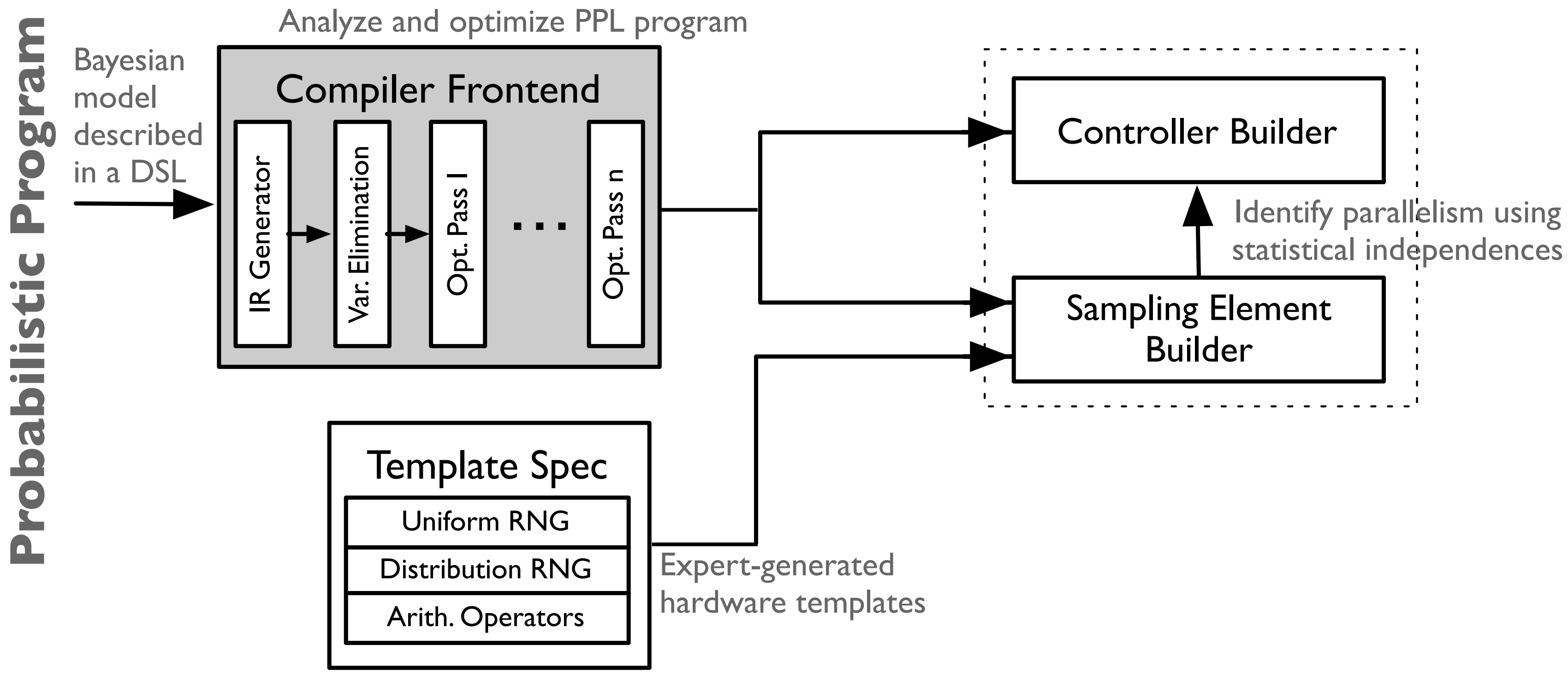
AcMC^2 Deep Dive

- Generic architecture for MCMC accelerators
  - Efficient high dimensional random number generators (samplers)
- Specializes architecture of accelerator given model, inference method



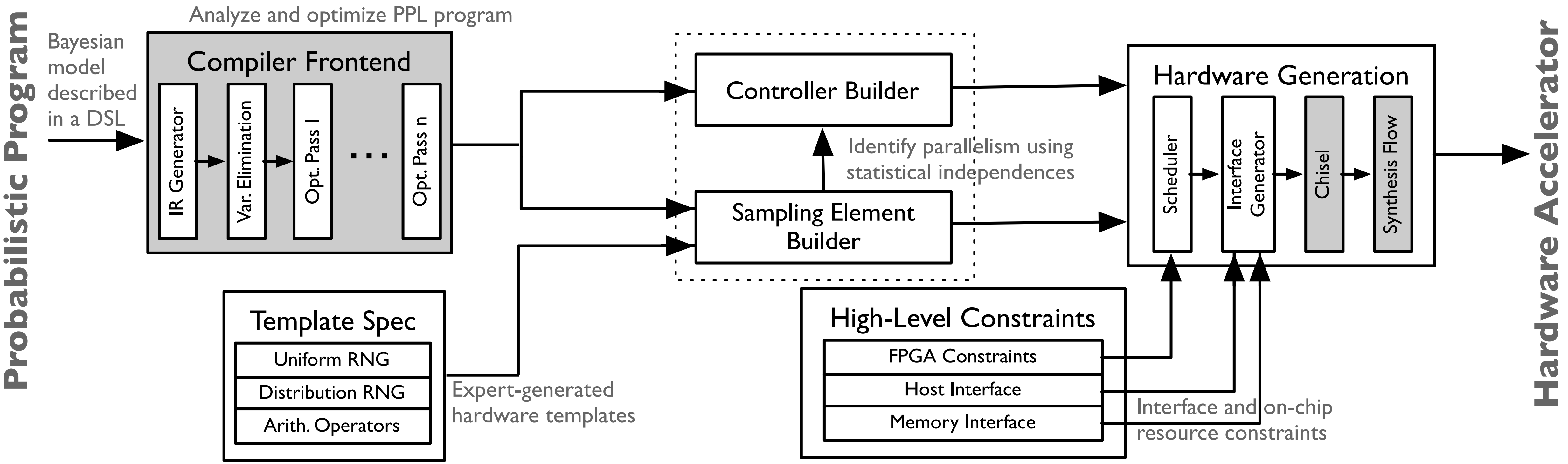
AcMC² Deep Dive

- Generic architecture for MCMC accelerators
  - Efficient high dimensional random number generators (samplers)
- Specializes architecture of accelerator given model, inference method



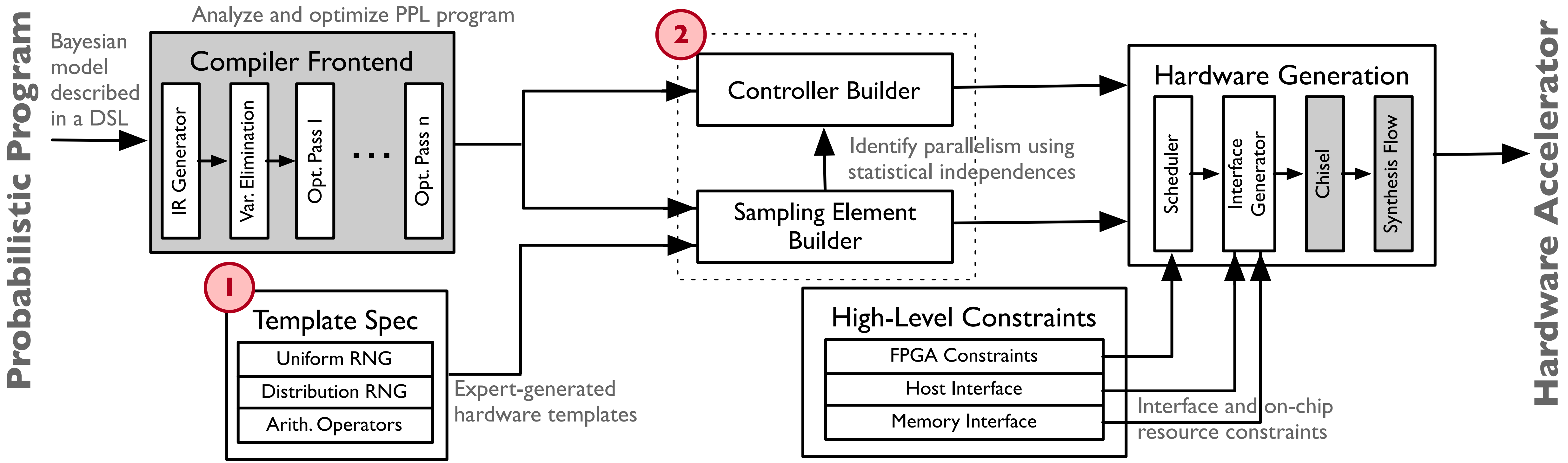
AcMC² Deep Dive

- Generic architecture for MCMC accelerators
  - Efficient high dimensional random number generators (samplers)
- Specializes architecture of accelerator given model, inference method

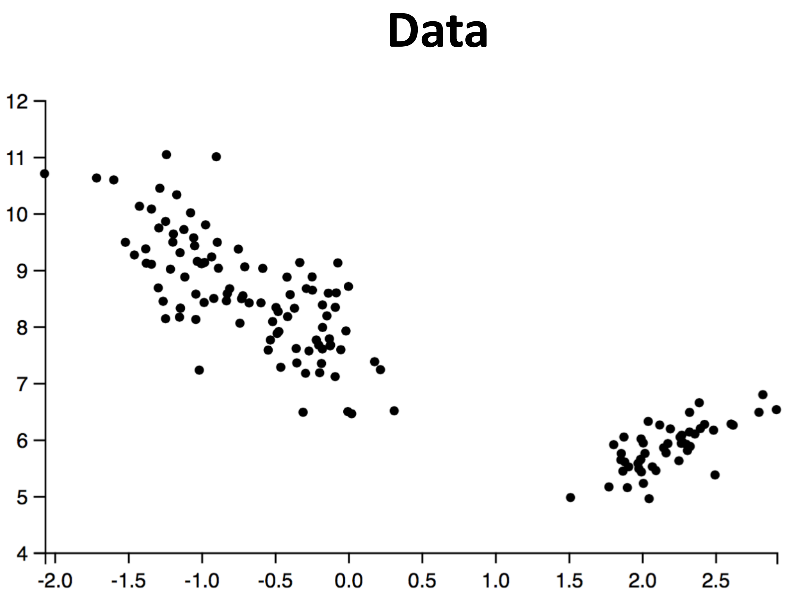


AcMC² Deep Dive

- Generic architecture for MCMC accelerators
  - Efficient high dimensional random number generators (samplers)
- Specializes architecture of accelerator given model, inference method

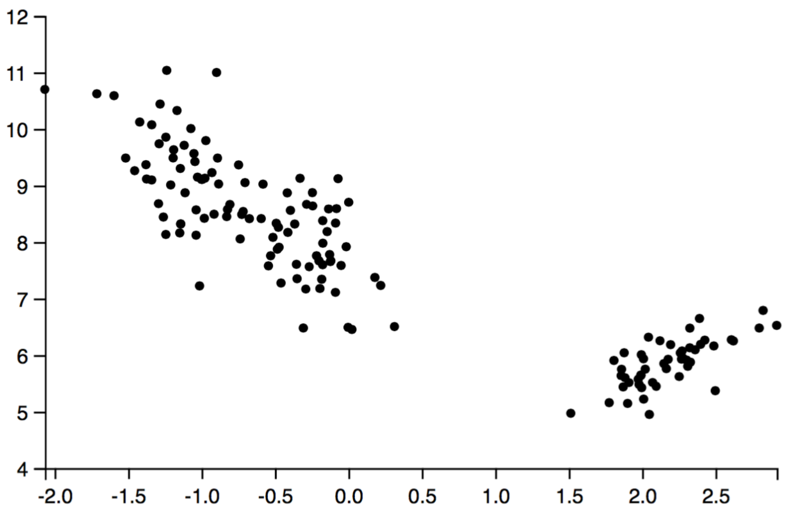


Example: GMM Clustering Data

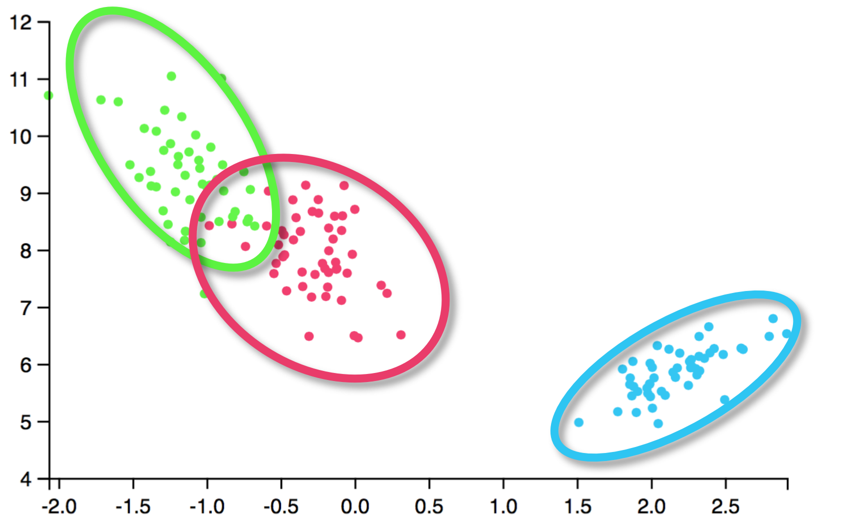


Example: GMM Clustering Data

Data

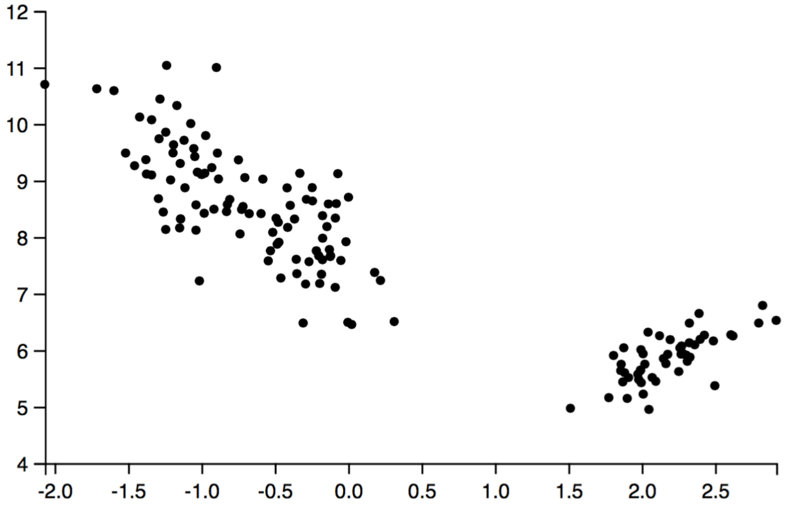


Inference

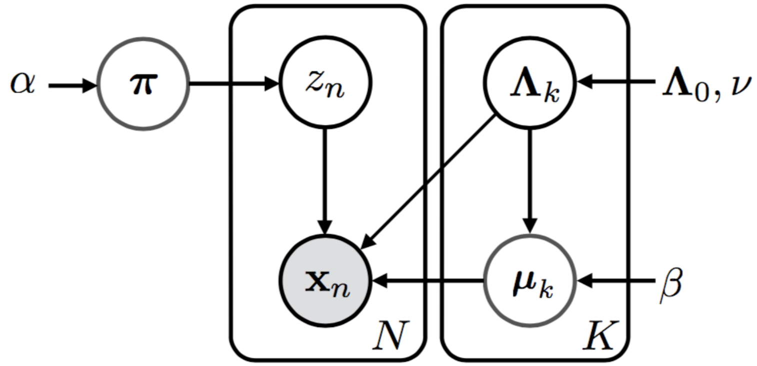


Example: GMM Clustering Data

Data



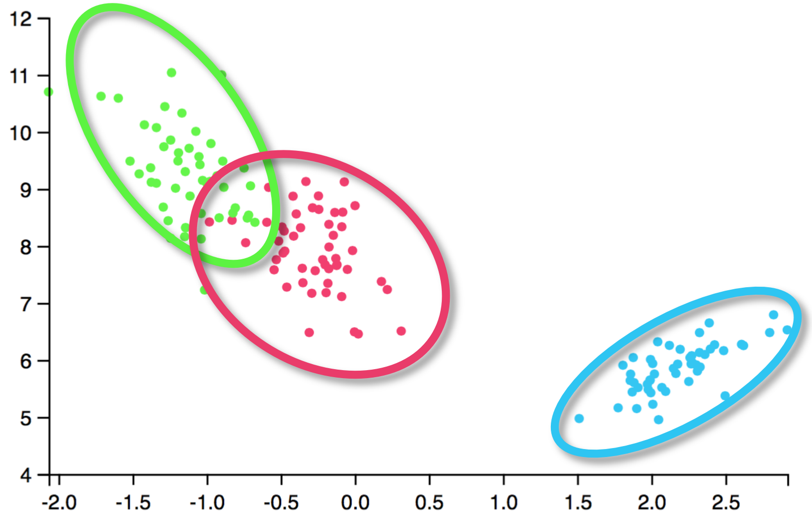
Generative Bayesian Model



$$\boldsymbol{\pi} \sim \text{Dirichlet}(\alpha)$$
$$\boldsymbol{\Lambda}_k \sim \text{Wishart}(\boldsymbol{\Lambda}_0, \nu)$$
$$\boldsymbol{\mu}_k | \boldsymbol{\Lambda}_k \sim \text{Normal}(\mathbf{0}, (\beta\boldsymbol{\Lambda}_k)^{-1})$$

$$z_n | \boldsymbol{\pi} \sim \text{Categorical}(\boldsymbol{\pi})$$
$$\mathbf{x}_n | z_n = k, \ \boldsymbol{\mu}_k, \ \boldsymbol{\Lambda}_k \sim \text{Normal}(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1}) \, .$$

Inference



Accelerable Kernels: Random Number Generators

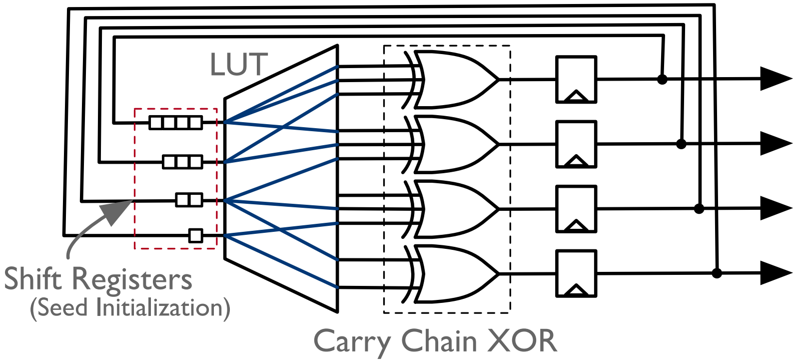
Fundamental operation used in MCMC: sampling uniform random numbers

Accelerable Kernels: Random Number Generators

Fundamental operation used in MCMC: sampling uniform random numbers

- Expert optimized FPGA URNG
  - 4bit LFSR
  - 1 cycle latency
  - 1 op/cycle

Single Primitive:
Uses single Logic Block for 4-bit RNG

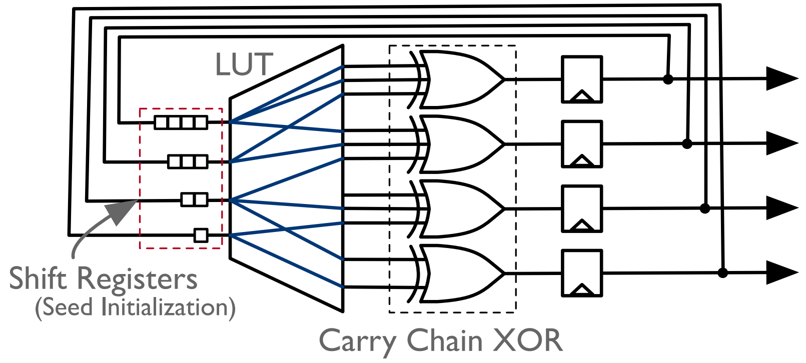


Accelerable Kernels: Random Number Generators

Fundamental operation used in MCMC: sampling uniform random numbers

- Expert optimized FPGA URNG
  - 4bit LFSR
  - 1 cycle latency
  - 1 op/cycle
- Traditional RNGs
  - Cryptographically Secure
  - Rejection sampling: Stalls

Single Primitive:
Uses single Logic Block for 4-bit RNG

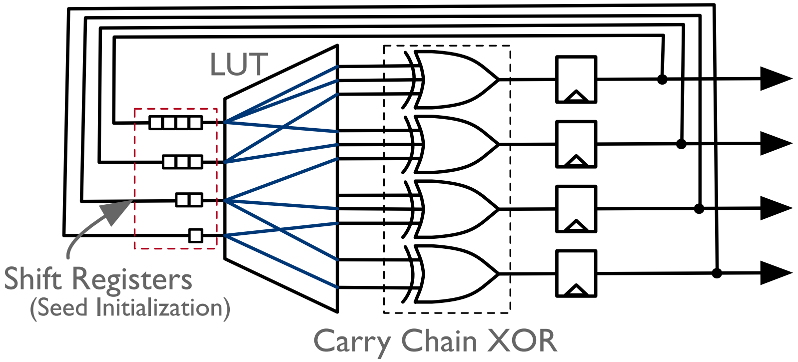


Accelerable Kernels: Random Number Generators

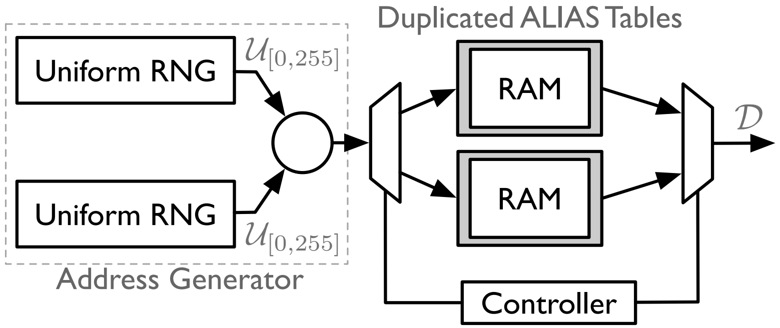
Fundamental operation used in MCMC: sampling uniform random numbers

- Expert optimized FPGA URNG
  - 4bit LFSR
  - 1 cycle latency
  - 1 op/cycle
- Traditional RNGs
  - Cryptographically Secure
  - Rejection sampling: Stalls

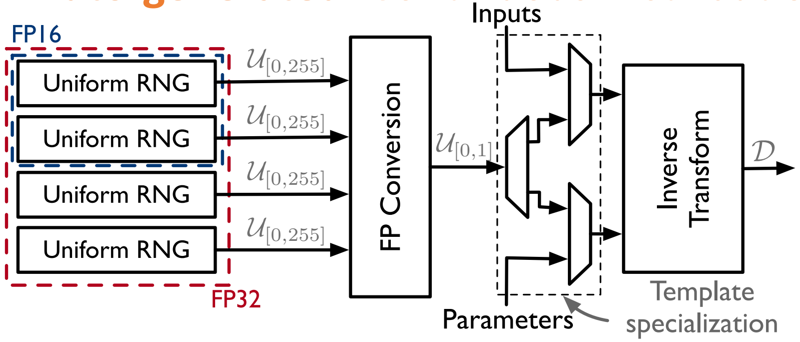
Single Primitive:
Uses single Logic Block for 4-bit RNG



Auto-generated: Discrete Distributions



Auto-generated: Continuous Distributions



Identifying Parallelism: Enter Markov Blankets

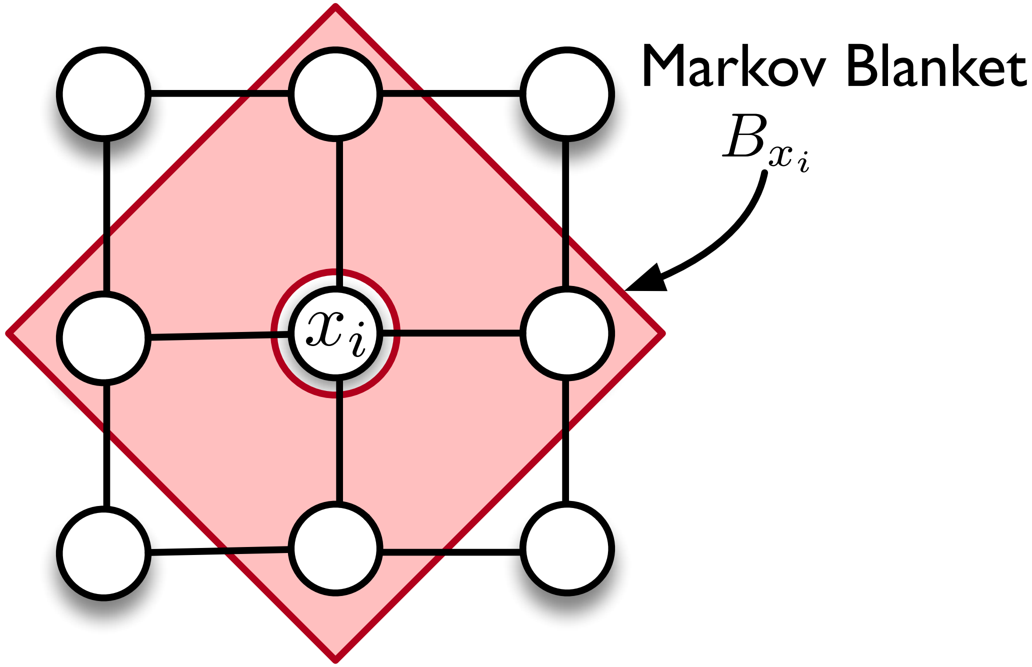
- How do we compose the samplers?
  - Program dataflow ordering is too conservative

Identifying Parallelism: Enter Markov Blankets

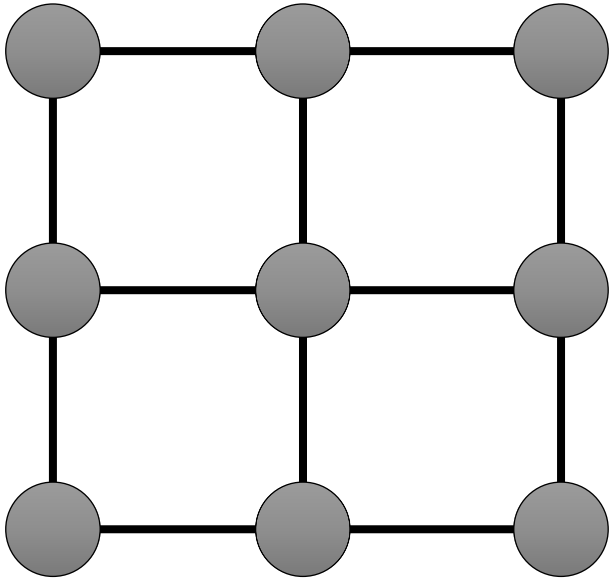
- How do we compose the samplers?
  - Program dataflow ordering is too conservative
- Use **conditional dependencies** to identify parallelism
- Set of nodes B_{x_i} for a node x_i such that:

$$\Pr(x_i | B_{x_i}, A) = \Pr(x_i | B_{x_i})$$

- **Markov blanket** is the only knowledge needed to predict behavior node.

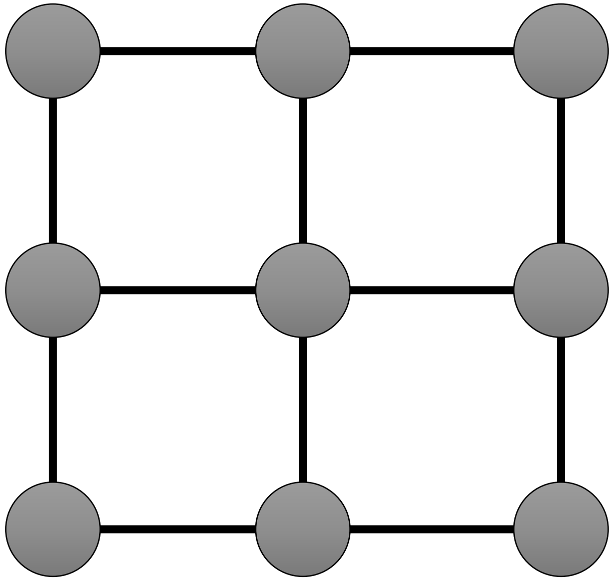


Identifying Parallelism: k-Colorings



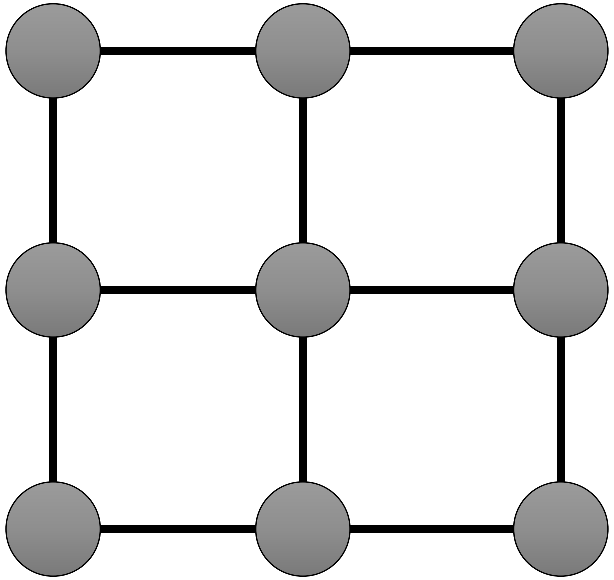
Identifying Parallelism: k-Colorings

- Compute a k-coloring of the graphical model



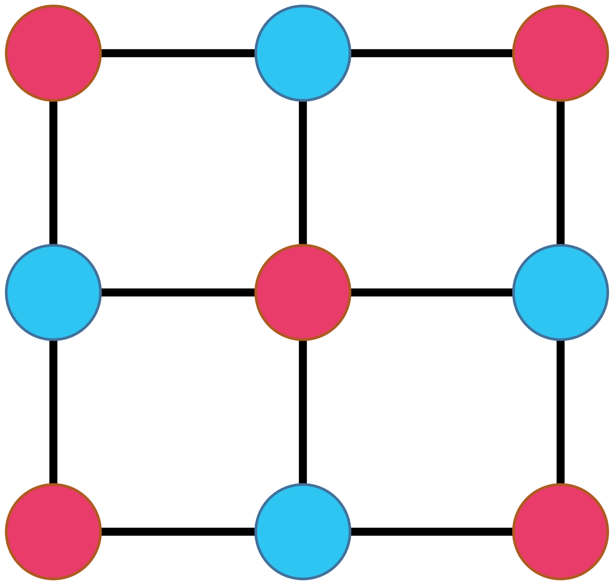
Identifying Parallelism: k-Colorings

- Compute a k-coloring of the graphical model
- Sample all variables with same color in parallel (Conditionally Independent)



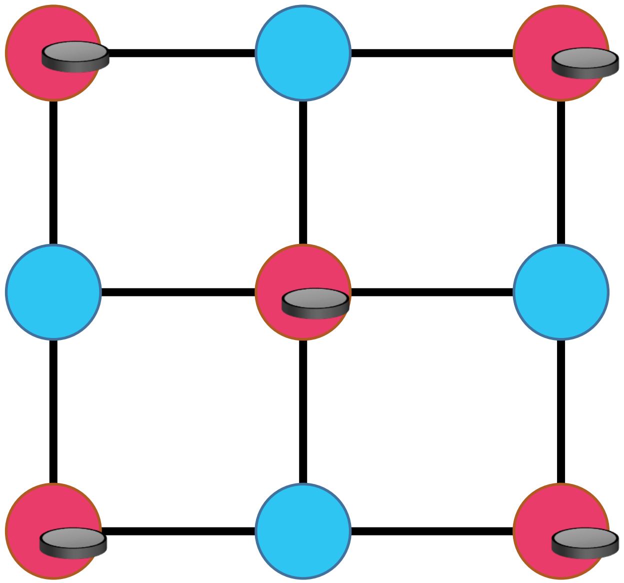
Identifying Parallelism: k-Colorings

- Compute a k-coloring of the graphical model
- Sample all variables with same color in parallel (Conditionally Independent)



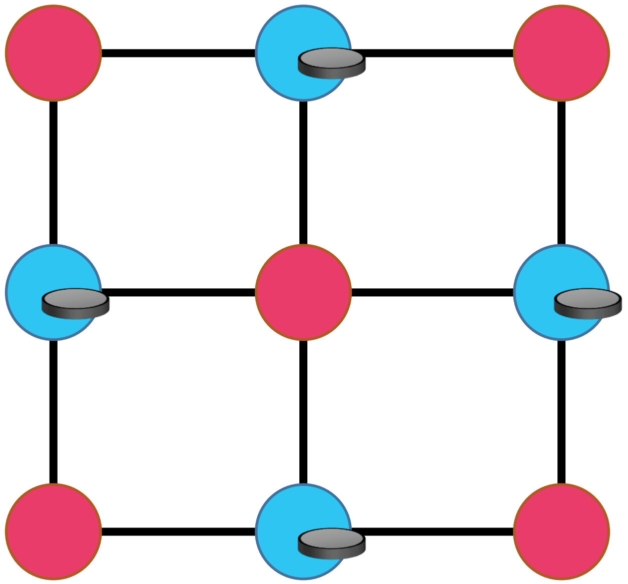
Identifying Parallelism: k-Colorings

- Compute a k-coloring of the graphical model
- Sample all variables with same color in parallel (Conditionally Independent)



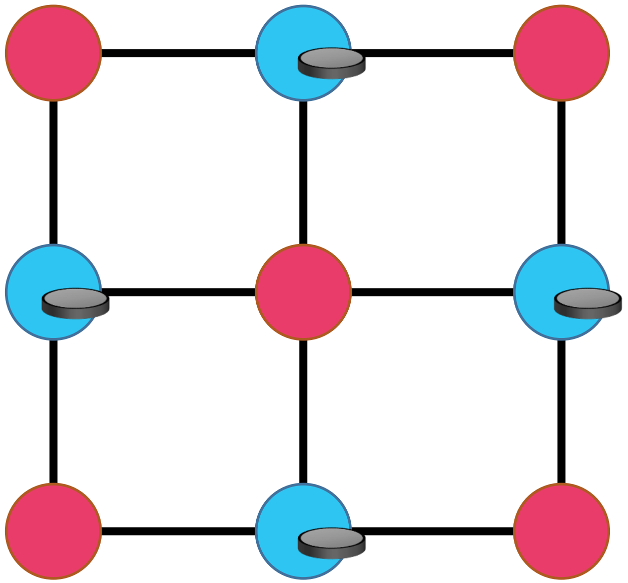
Identifying Parallelism: k-Colorings

- Compute a k-coloring of the graphical model
- Sample all variables with same color in parallel (Conditionally Independent)



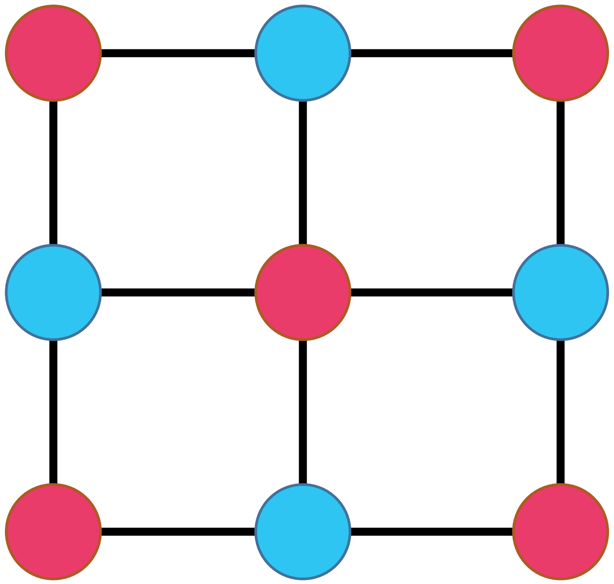
Identifying Parallelism: k-Colorings

- Compute a k-coloring of the graphical model
- Sample all variables with same color in parallel (Conditionally Independent)
- Synthesize state machines corresponding to coloring



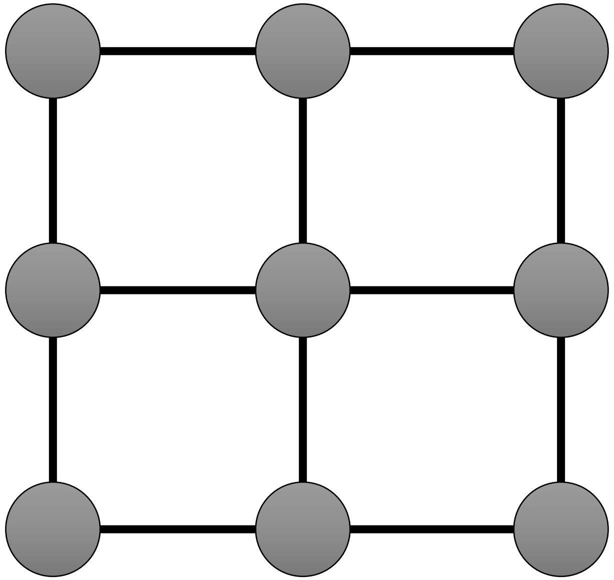
Identifying Parallelism: k-Colorings

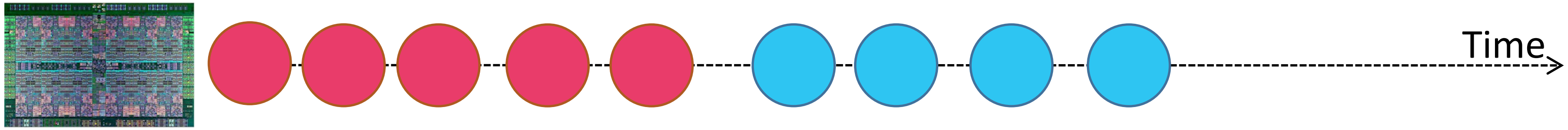
- Compute a k-coloring of the graphical model
- Sample all variables with same color in parallel (Conditionally Independent)
- Synthesize state machines corresponding to coloring



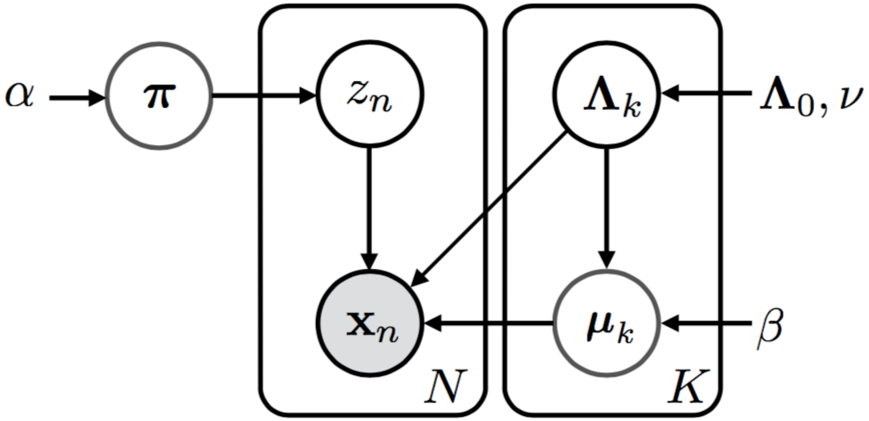
Identifying Parallelism: k-Colorings

- Compute a k-coloring of the graphical model
- Sample all variables with same color in parallel (Conditionally Independent)
- Synthesize state machines corresponding to coloring
- Equivalent to sequential

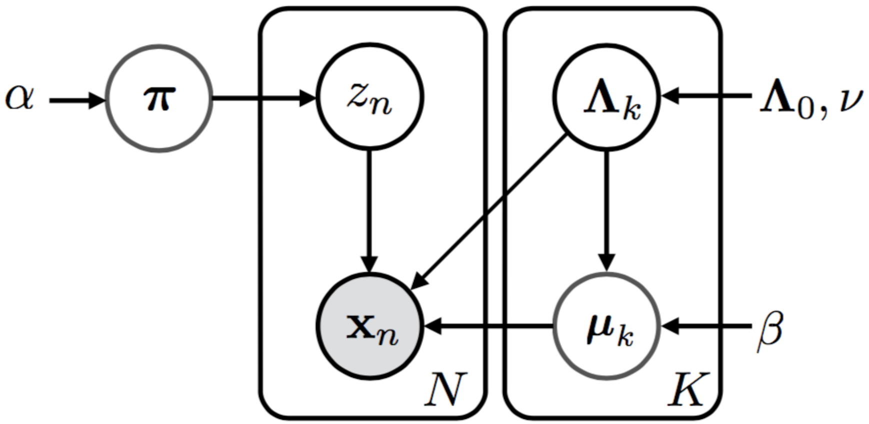




Parallelism in the GMM Clustering Example



Parallelism in the GMM Clustering Example



$\Pr(\alpha|\pi)$

$\Pr(\pi|\alpha, z_n)$

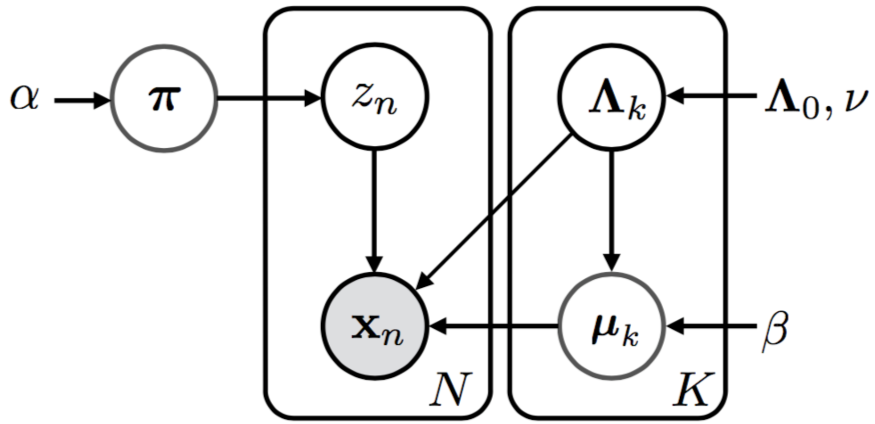
$\Pr(z_n|\pi, x_n, \Lambda_k, \mu_k)$

$\Pr(x_n|z_n, \Lambda_k, \mu_k)$

$\Pr(\mu_k|\beta, \Lambda_k, x_n)$

$\Pr(\Lambda_k|\Lambda_0, \mu_k, x_n, z_n)$

Parallelism in the GMM Clustering Example



$\Pr(\alpha|\pi)$

$\Pr(\pi|\alpha, z_n)$

$\Pr(z_n|\pi, x_n, \Lambda_k, \mu_k)$

$\Pr(x_n|z_n, \Lambda_k, \mu_k)$

$\Pr(\mu_k|\beta, \Lambda_k, x_n)$

$\Pr(\Lambda_k|\Lambda_0, \mu_k, x_n, z_n)$

Revisiting the GMM Clustering Example

$\Pr(\alpha|\pi)$

$\Pr(z_n|\pi, x_n, \Lambda_k, \mu_k)$

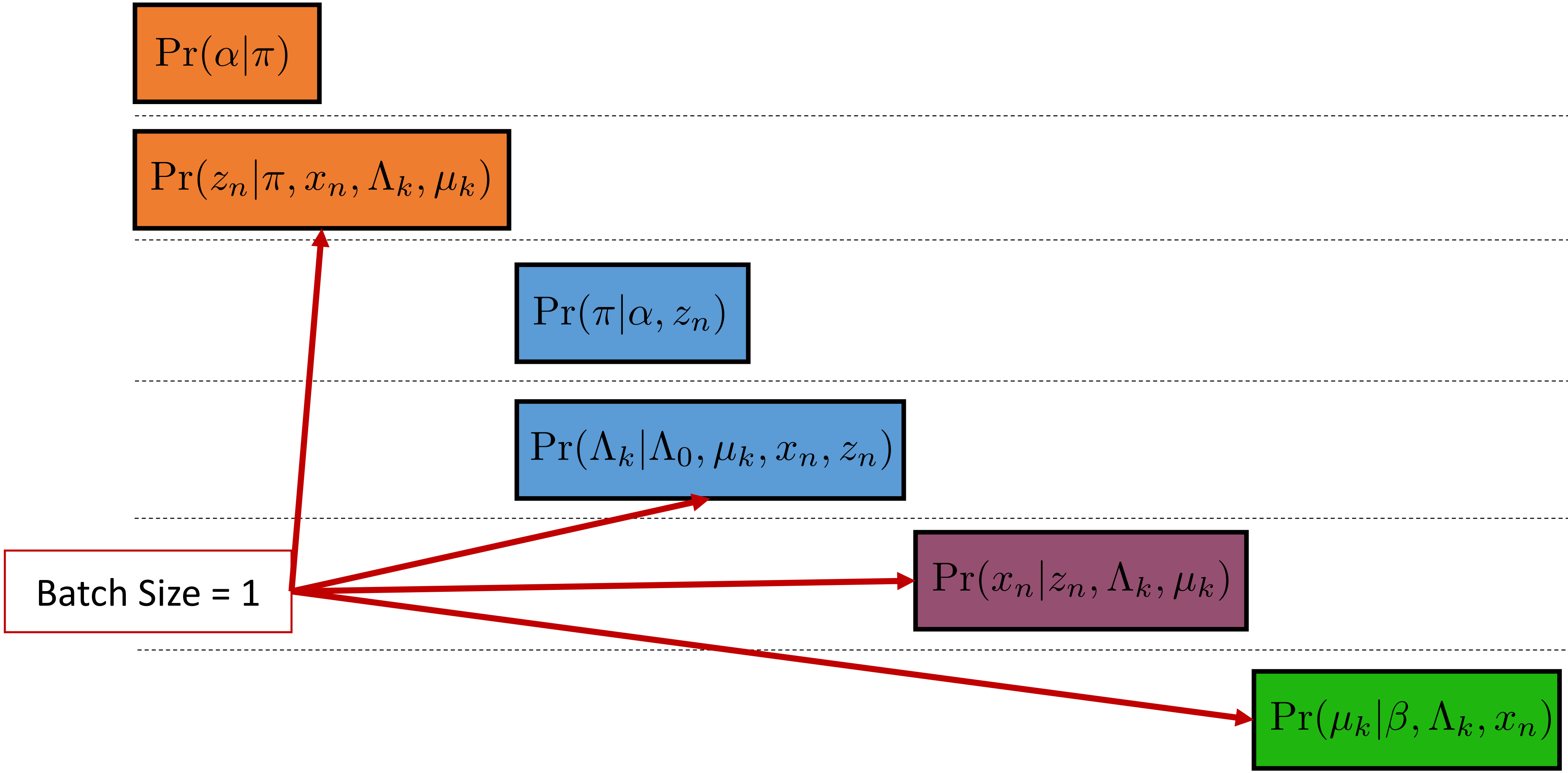
$\Pr(\pi|\alpha, z_n)$

$\Pr(\Lambda_k|\Lambda_0, \mu_k, x_n, z_n)$

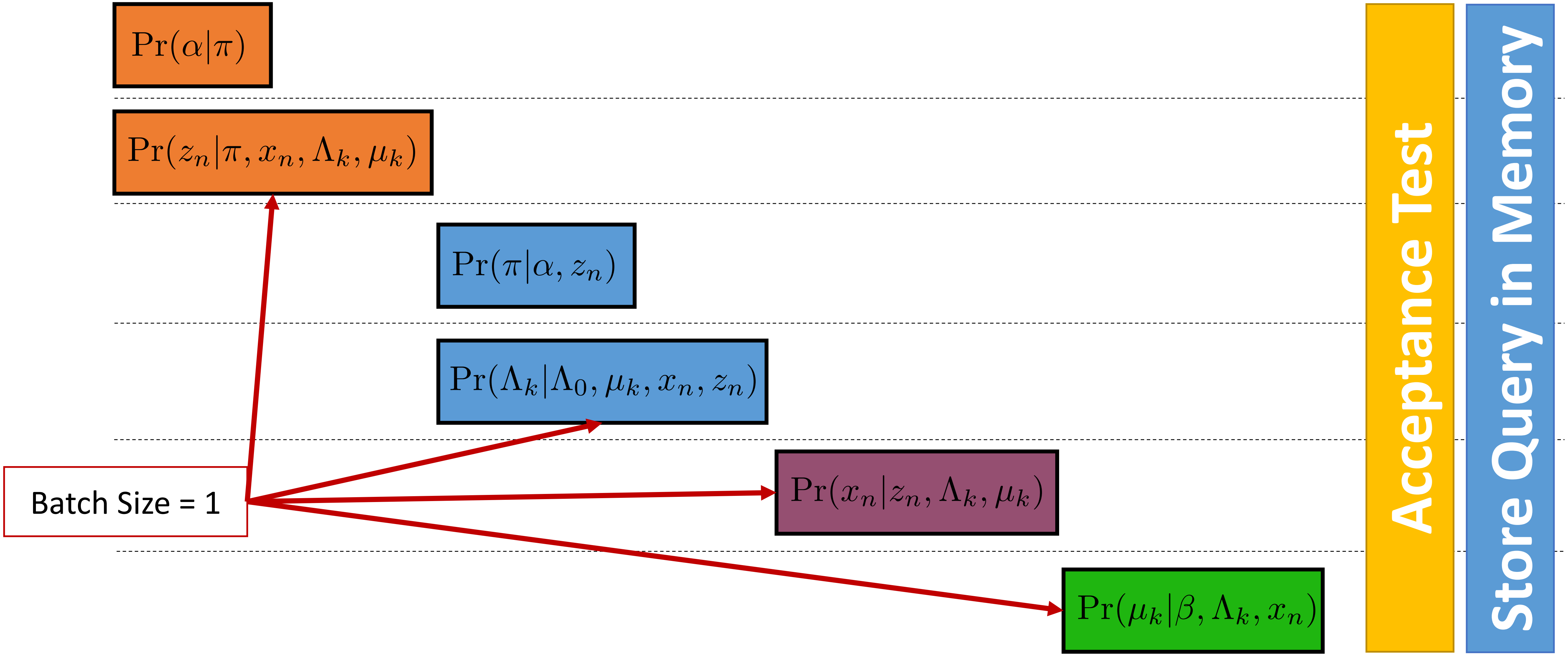
$\Pr(x_n|z_n, \Lambda_k, \mu_k)$

$\Pr(\mu_k|\beta, \Lambda_k, x_n)$

Revisiting the GMM Clustering Example



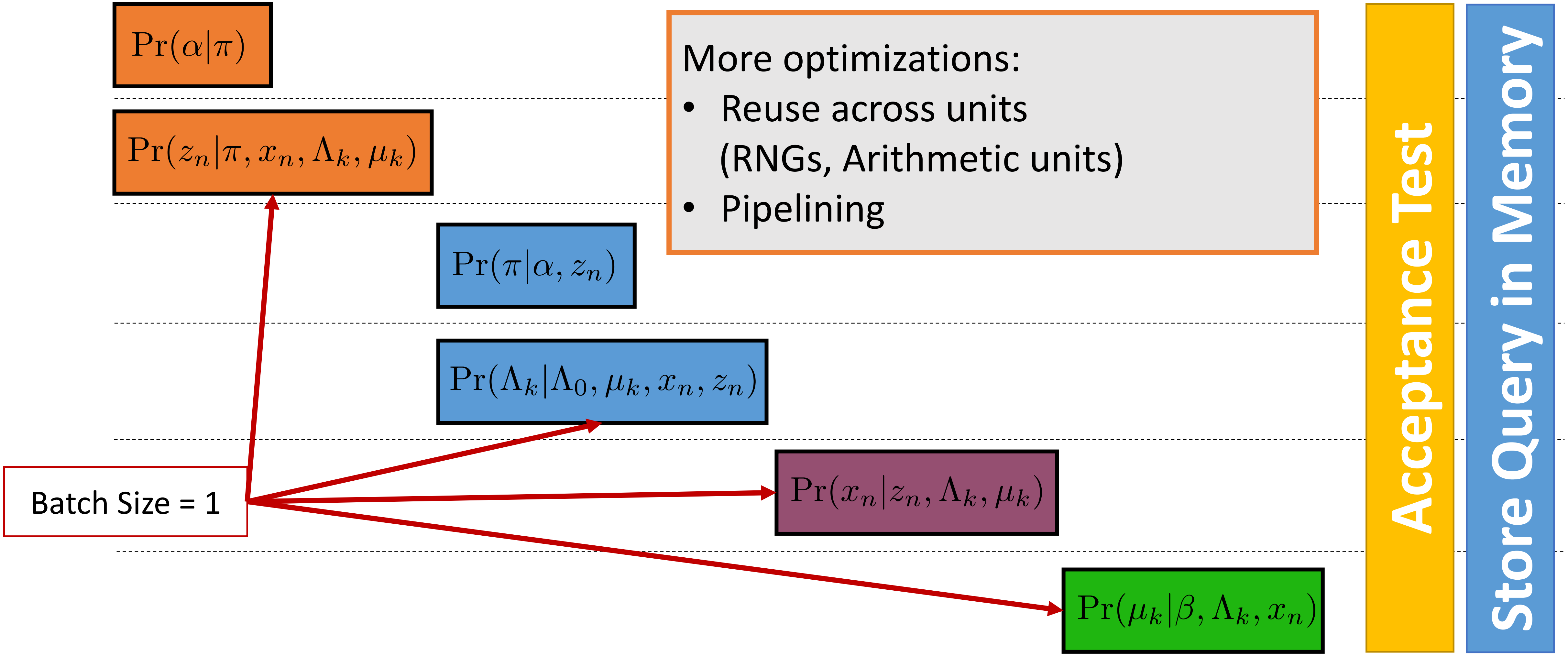
Revisiting the GMM Clustering Example



Revisiting the GMM Clustering Example

More optimizations:

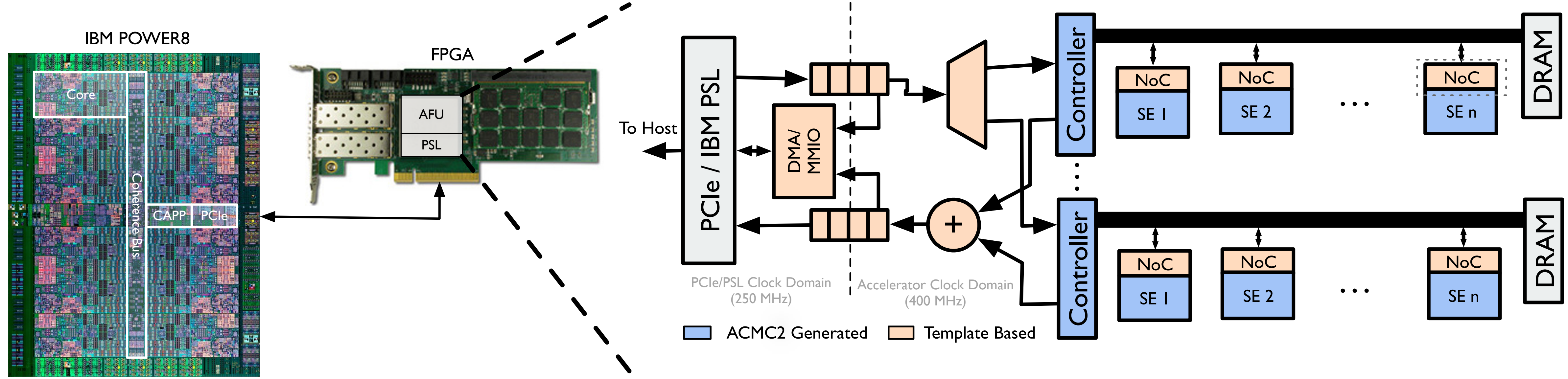
- Reuse across units (RNGs, Arithmetic units)
- Pipelining



Other Details in the Paper

- Compositional MCMC
  - Gibbs, Metropolis Hastings, Hamiltonian
- Speculative Execution
  - Speculate past rejected samples
- Accuracy – Performance Tradeoffs
  - Bloom Filters; Precision
- Generating IBM-CAPI based DMA Engine
  - Little’s Law

Implementation



6-core IBM POWER8

CAPI attached Virtex 7 FPGA

Sampling Element (SE) = RNGs + k-coloring controller

N x M sampling elements
N = 4 (4 DRAM channels on FPGA board)
M = max that can be fit on FPGA

Evaluation: AcMC^2 in Real World Models

Epilepsy/Neurosicnece: Identifying epilepsy affected brain regions [Varatharajah, NeurIPS17]

Security: Preempting advanced persistent threats using host/network IDSs [Cao, HOTSOS15]

Evaluation: AcMC^2 in Real World Models

Epilepsy/Neurosicnece: Identifying epilepsy affected brain regions [Varatharajah, NeurIPS17]

Security: Preempting advanced persistent threats using host/network IDSs [Cao, HOTSOS15]

Embedded Medical Devices

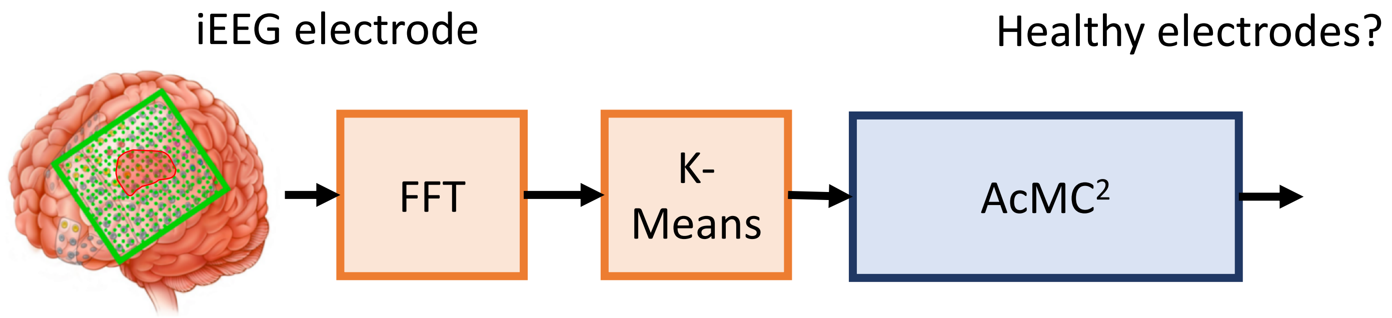
Datacenter network monitoring tools

Evaluation: AcMC^2 in Real World Models

Epilepsy/Neurosicnece: Identifying epilepsy affected brain regions [Varatharajah, NeurIPS17]

Security: Preempting advanced persistent threats using host/network IDSs [Cao, HOTSOS15]

Embedded Medical Devices ⟷ **Datacenter network monitoring tools**

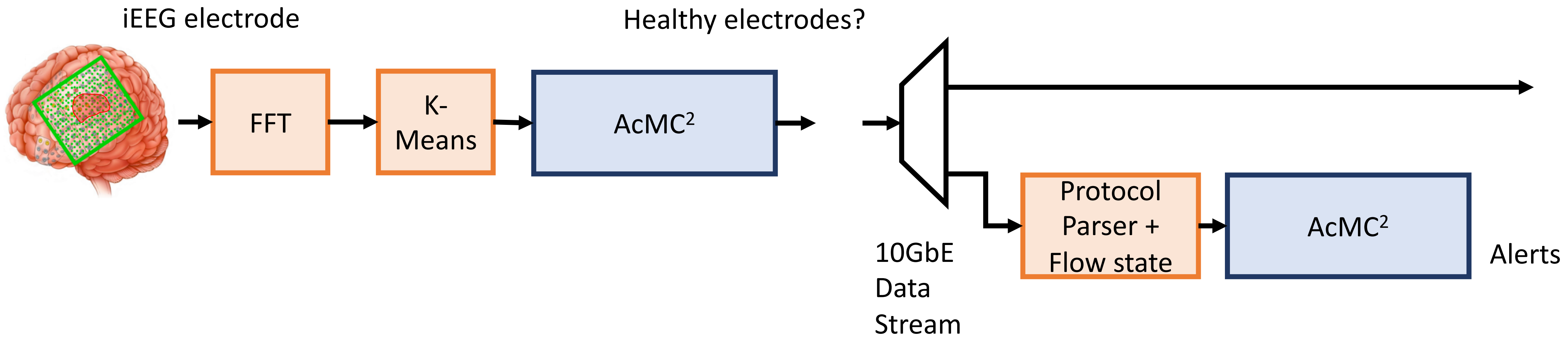


Evaluation: AcMC^2 in Real World Models

Epilepsy/Neurosicnece: Identifying epilepsy affected brain regions [Varatharajah, NeurIPS17]

Security: Preempting advanced persistent threats using host/network IDSs [Cao, HOTSOS15]

Embedded Medical Devices ⟷ **Datacenter network monitoring tools**



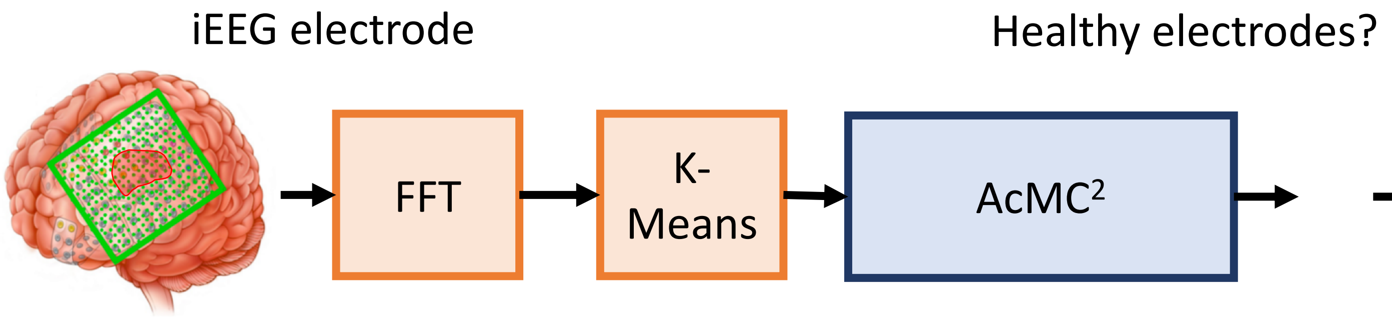
Evaluation: AcMC² in Real World Models

Epilepsy/Neurosicnece: Identifying epilepsy affected brain regions [Varatharajah, NeurIPS17]

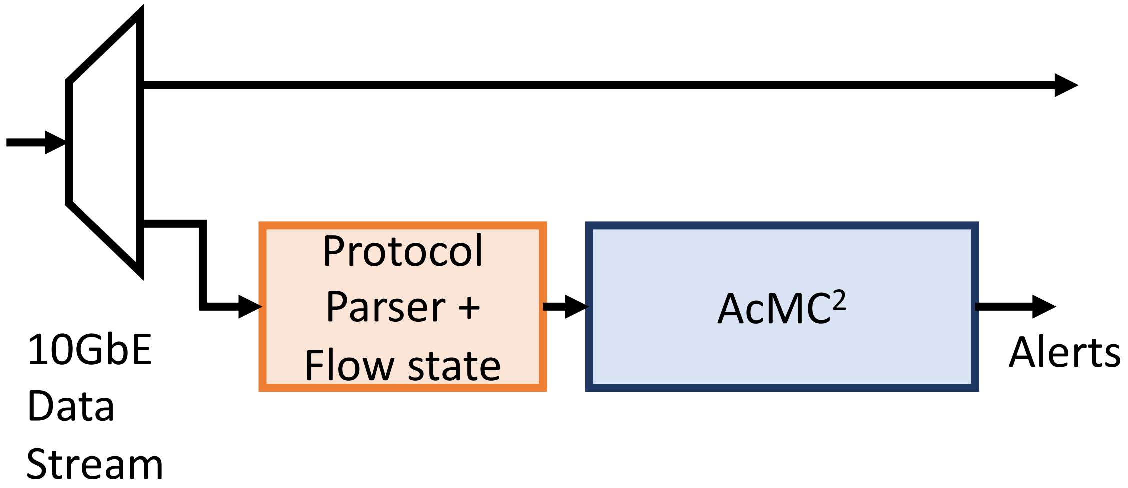
Security: Preempting advanced persistent threats using host/network IDSs [Cao, HOTSOS15]

Embedded Medical Devices

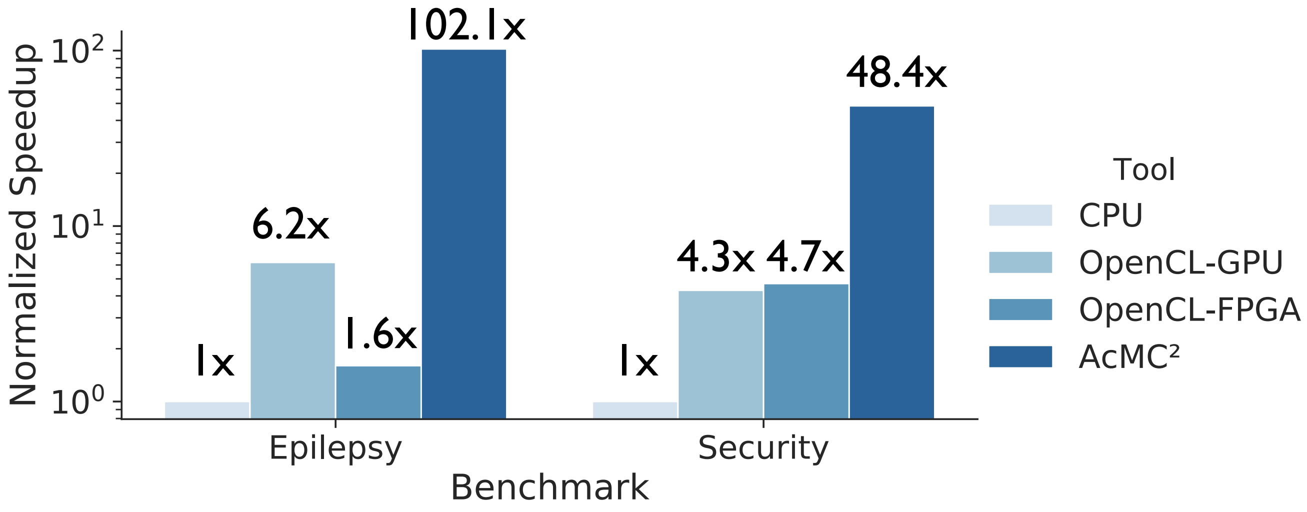
Datacenter network monitoring tools



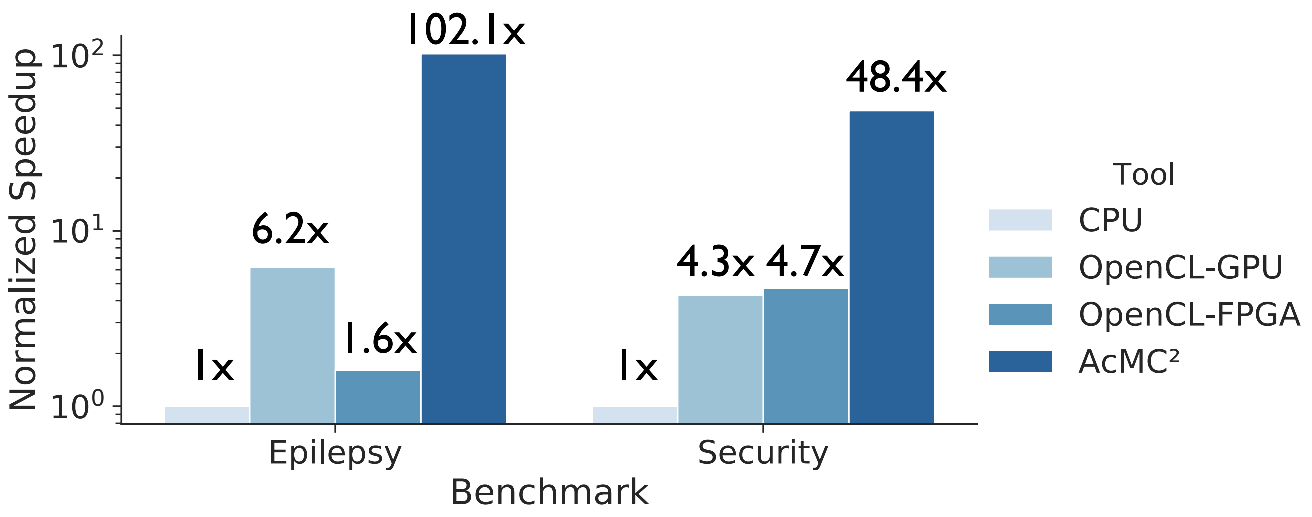
Inference accounts for >90% of the latency

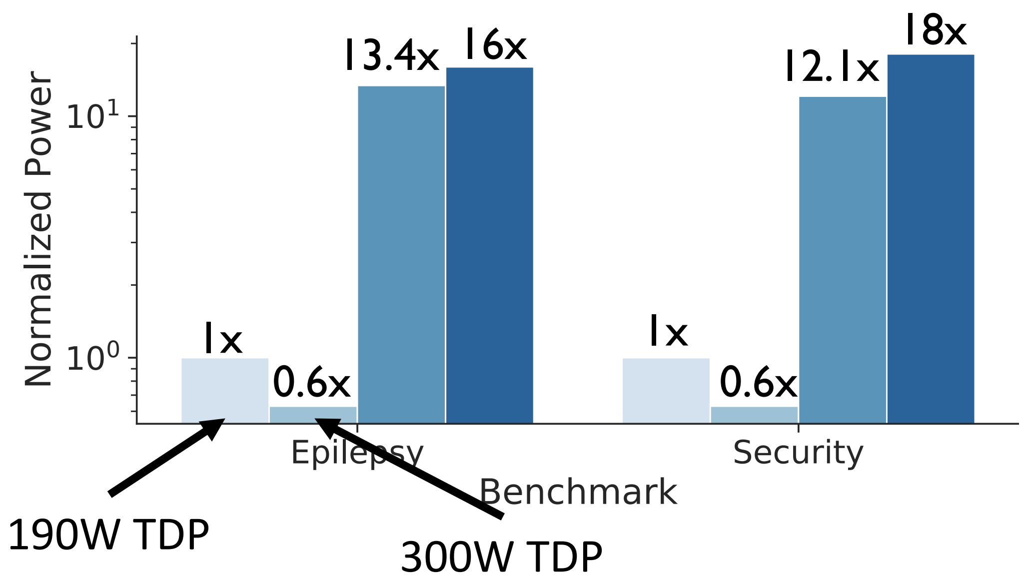


Results: Real World Case Studies

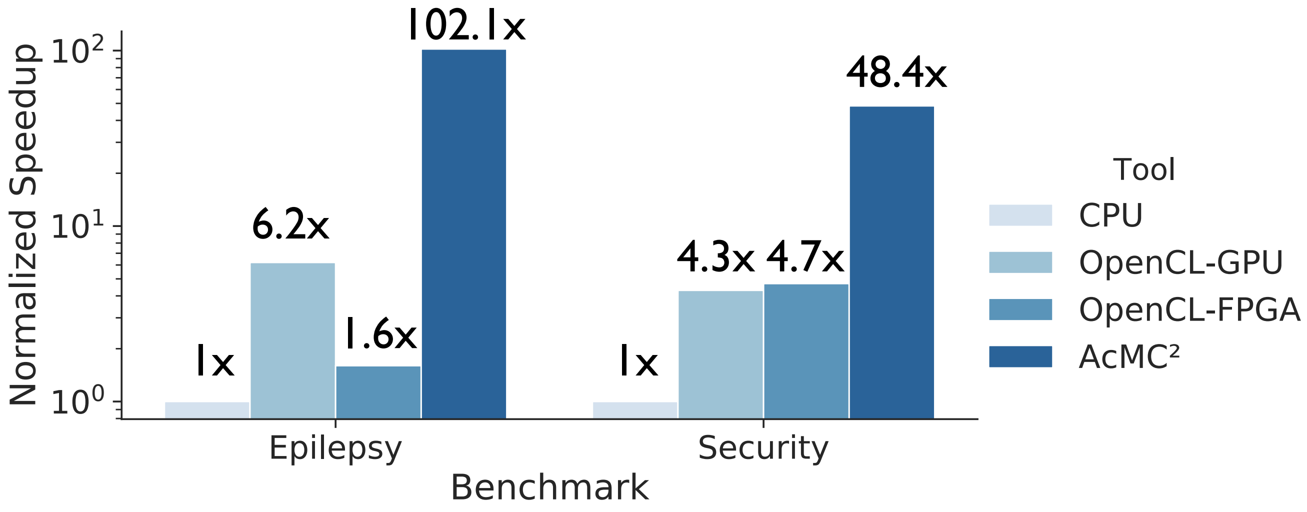


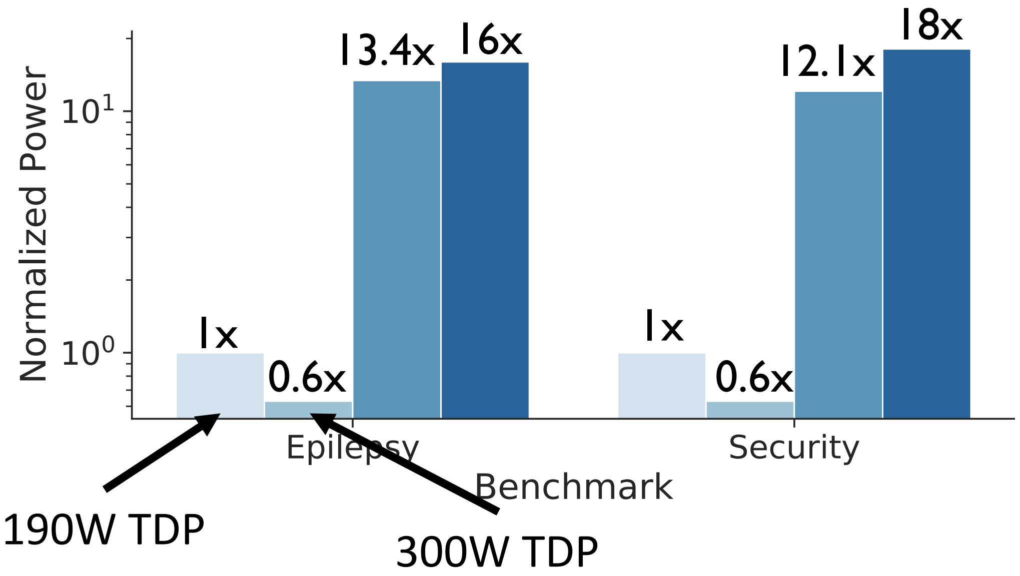
Results: Real World Case Studies





Results: Real World Case Studies





Significantly better results at much simpler code complexity
LoC AcMC² – 183 for C1 & 146 for C2
LoC OpenCL – 961 for C1 & 4861 for C2

Conclusion

- AcMC^2: A High Level Synthesis Compiler for Probabilistic Programs

- Code is open-source and available at https://gitlab.engr.Illinois.edu/DEPEND/AcMC2

Conclusion

- AcMC^2: A High Level Synthesis Compiler for Probabilistic Programs
- Code is open-source and available at https://gitlab.engr.Illinois.edu/DEPEND/AcMC2
- Looking forward
  - How does these models fit in the context of Deep Learning? – Bayesian Deep Learning