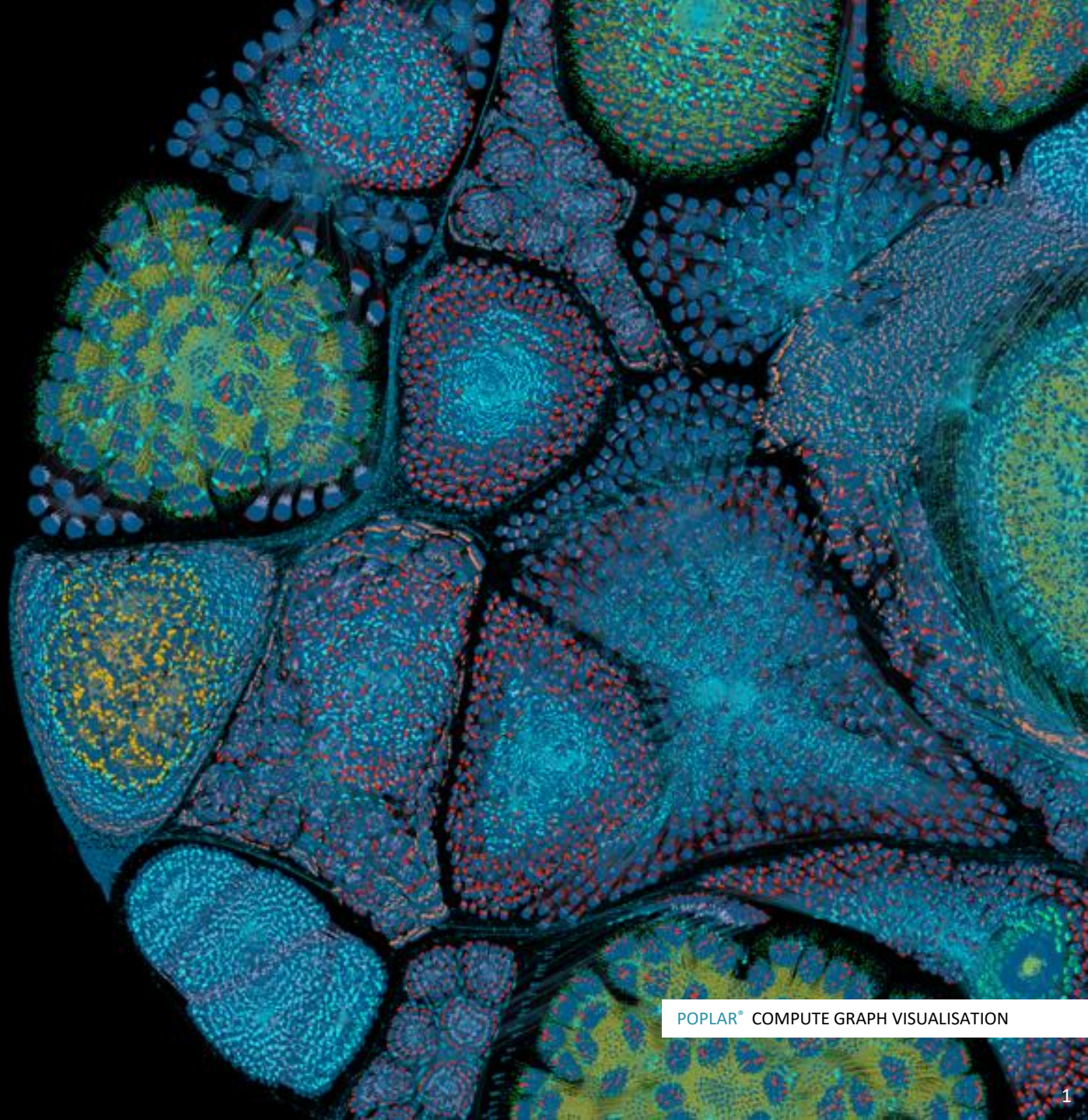


'MODEL' MARRIAGES: PAIRING IMPLEMENTATIONS & HARDWARE

Luke Markham
ML Engineer - Graphcore



POPLAR® COMPUTE GRAPH VISUALISATION

Optimizing model runtimes



Imagine...

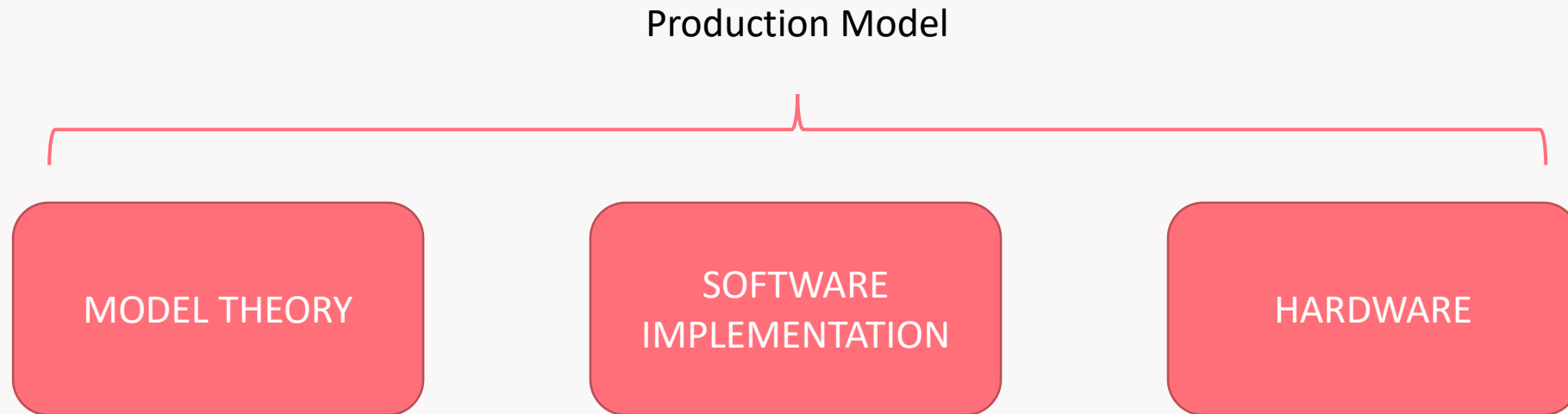
-> You're running a quant team

-> You've validated a model

-> It's too slow to run in production

-> You need to optimize its runtime

(RUNTIME RELEVANT) INGREDIENTS OF A PRODUCTION MODEL

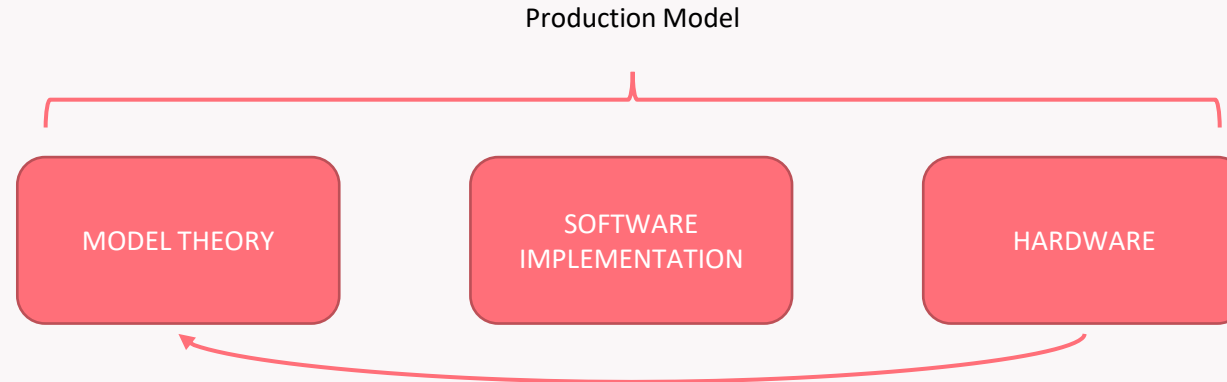


Not included:

DATA



(RUNTIME RELEVANT) INGREDIENTS OF A PRODUCTION MODEL



The Hardware Lottery

Sara Hooker

Google Research, Brain Team
shooker@google.com

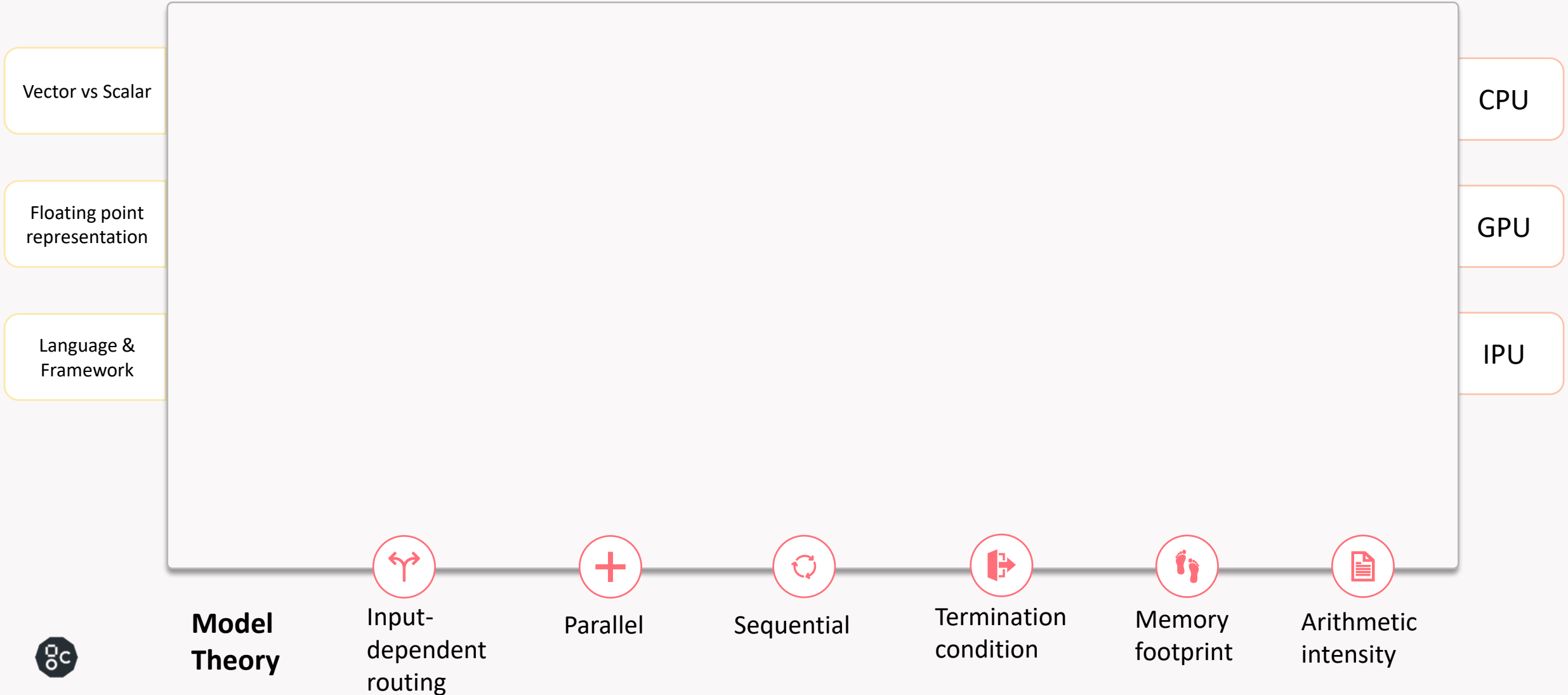
Abstract

Hardware, systems and algorithms research communities have historically had different incentive structures and fluctuating motivation to engage with each other explicitly. This historical treatment is odd given that hardware and software have frequently determined which research ideas succeed (and fail). This essay introduces the term hardware lottery to describe when a research idea wins because it is suited to the available software and hardware and *not* because the idea is superior to alternative research directions. Examples from early computer science history illustrate how hardware lotteries can delay research progress by casting successful ideas as failures. These lessons are particularly salient given the advent of domain specialized hardware which make it increasingly costly to stray off of the beaten path of research ideas. This essay posits that the gains from progress in computing are likely to become even more uneven, with certain research directions moving into the fast-lane while progress on others is further obstructed.

A MODEL MARRIAGE

**Software
Implementation**

Hardware



A MODEL MARRIAGE

**Software
Implementation**

Hardware

Vector vs Scalar

Floating point
representation

Language &
Framework

Tree-based workloads,
or conditional control
flow (if-statements).
Decision tree is a good
example.

CPU

GPU

IPU



Input-
dependent
routing



Parallel



Sequential



Termination
condition



Memory
footprint



Arithmetic
intensity

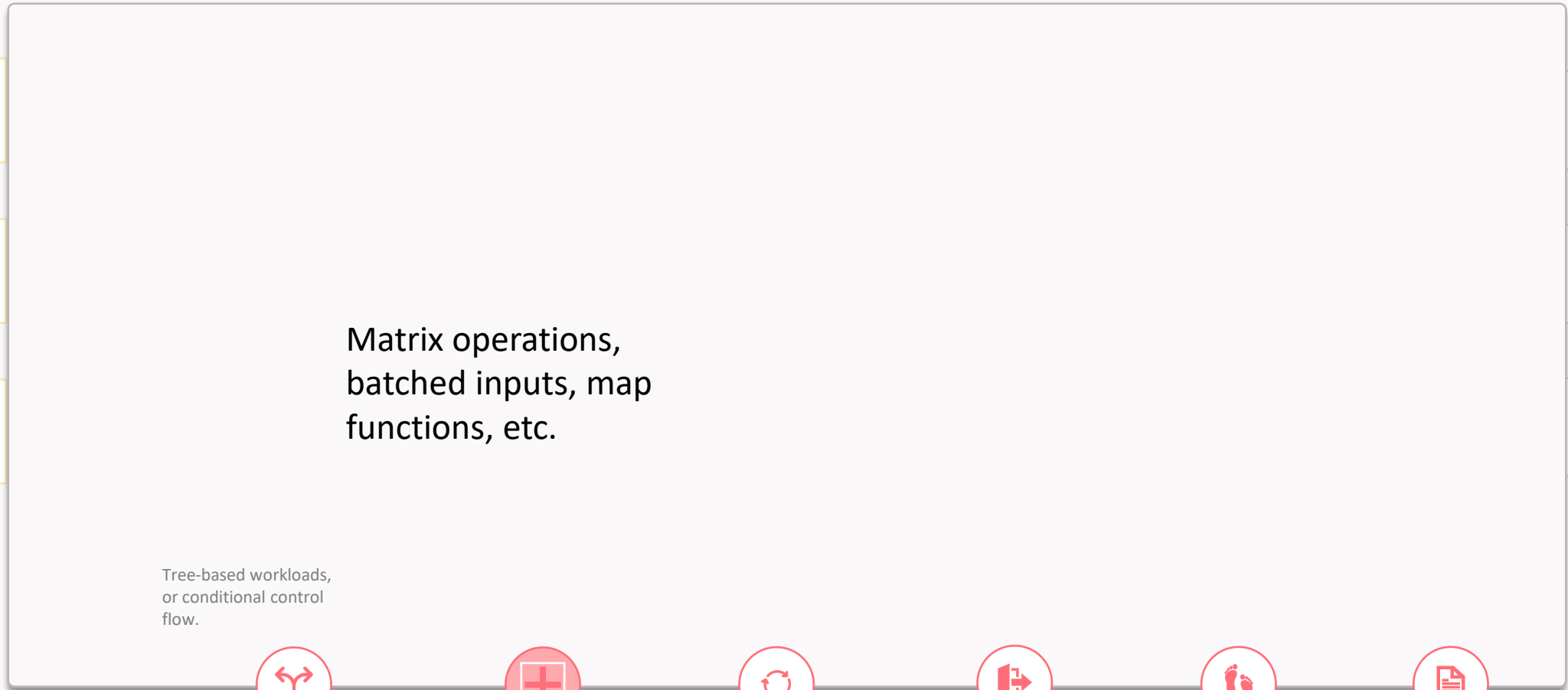


**Model
Theory**

A MODEL MARRIAGE

**Software
Implementation**

Hardware



**Model
Theory**



Input-
dependent
routing



Parallel



Sequential



Termination
condition



Memory
footprint



Arithmetic
intensity



A MODEL MARRIAGE

**Software
Implementation**

Hardware

Vector vs Scalar

Floating point
representation

Language &
Framework

CPU

GPU

IPU

Iterative algorithms, loops,
recursive algorithms.

Tree-based workloads,
or conditional control
flow.

Matrix operations,
batched inputs,
map functions, etc.



**Model
Theory**
Input-
dependent
routing



Parallel



Sequential



Termination
condition



Memory
footprint



Arithmetic
intensity



A MODEL MARRIAGE

Software
Implementation

Hardware

Vector vs Scalar

Floating point
representation

Language &
Framework

CPU

GPU

IPU

Fixed sequential steps vs
convergence vs other
condition?

Tree-based workloads,
or conditional control
flow.

Matrix operations,
batched inputs,
map functions, etc.

Iterative algorithms,
loops, recursive
algorithms.



**Model
Theory**

Input-
dependent
routing

Parallel

Sequential

Termination
condition

Memory
footprint

Arithmetic
intensity



A MODEL MARRIAGE

Software
Implementation

Hardware

Vector vs Scalar

Floating point
representation

Language &
Framework

CPU

GPU

IPU

How does it compare to
on-chip and off-chip?

Tree-based workloads,
or conditional control
flow.

Matrix operations,
batched inputs,
map functions, etc.

Iterative algorithms,
loops, recursive
algorithms.

Fixed sequential steps vs
convergence vs other
condition?



**Model
Theory**

Input-
dependent
routing

Parallel

Sequential

Termination
condition

Memory
footprint

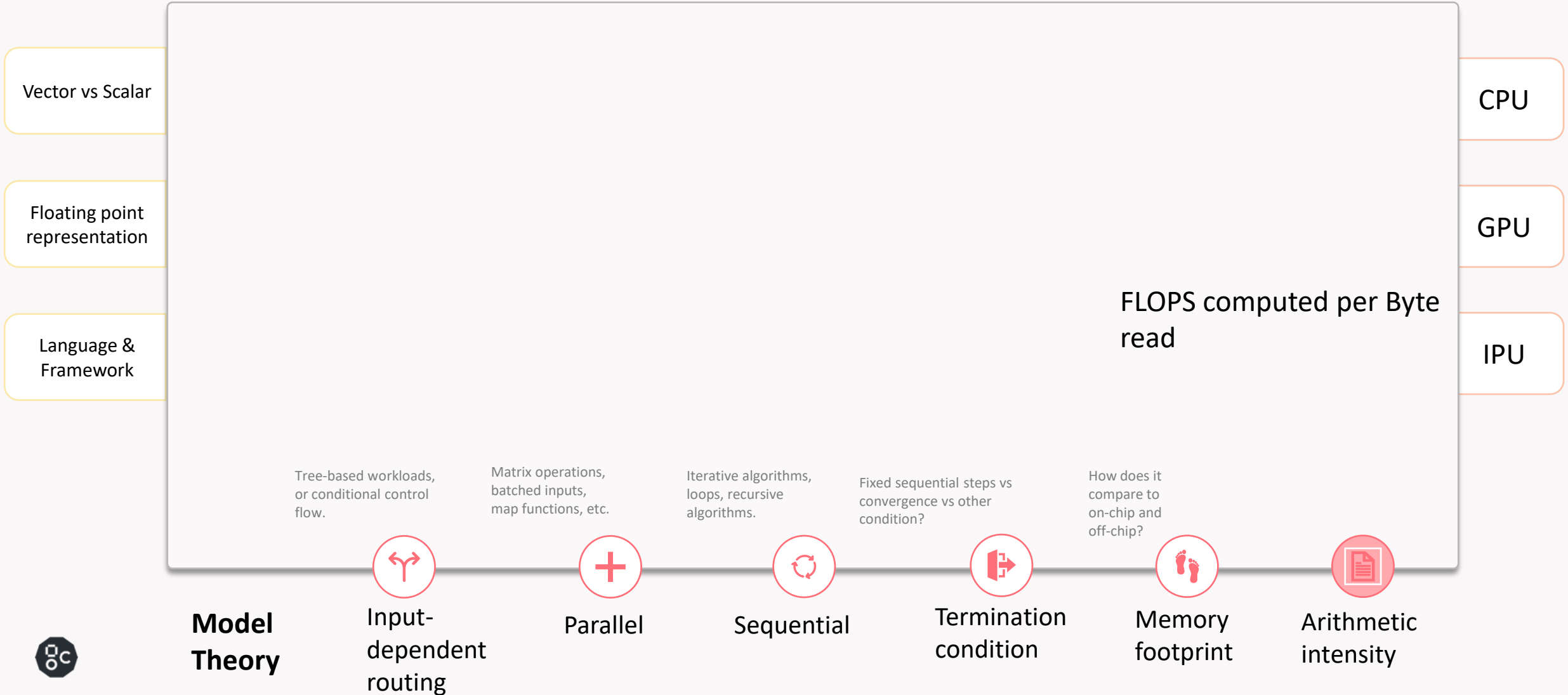
Arithmetic
intensity



A MODEL MARRIAGE

Software
Implementation

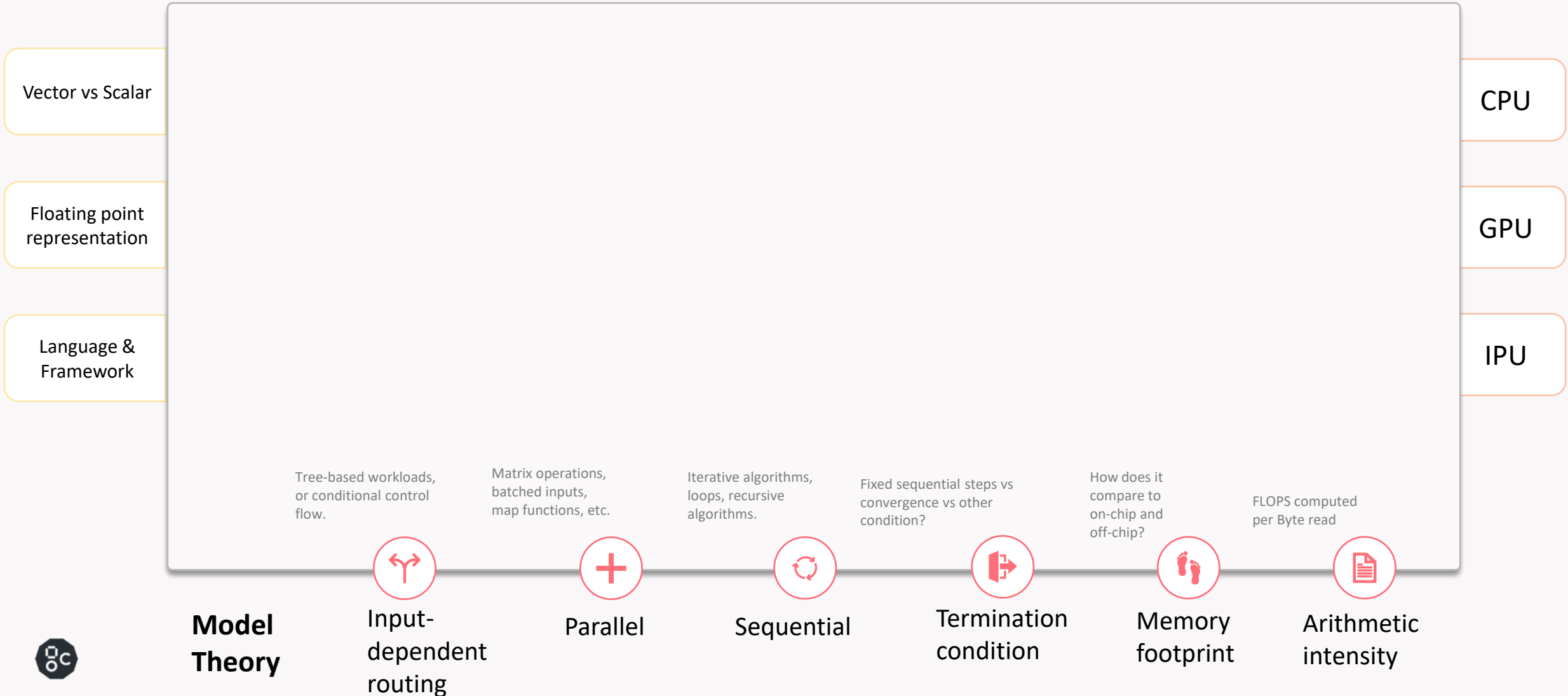
Hardware



A MODEL MARRIAGE

Software
Implementation

Hardware

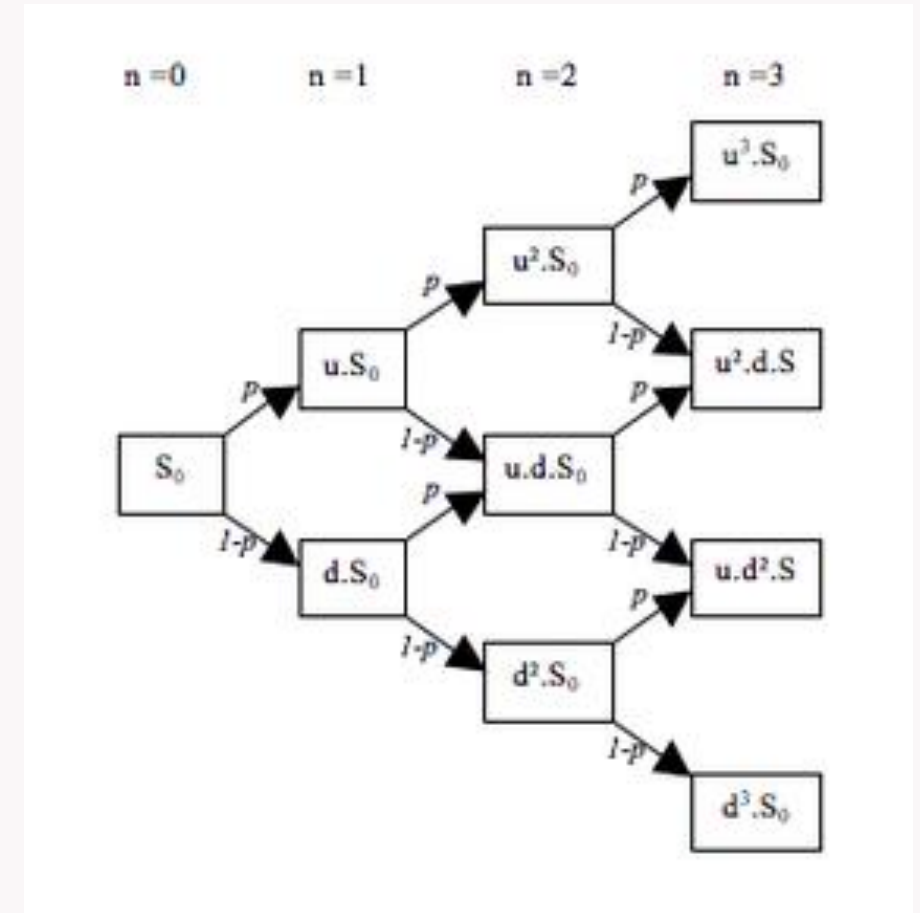


CASE STUDY



CRR ALGORITHM – AN OVERVIEW

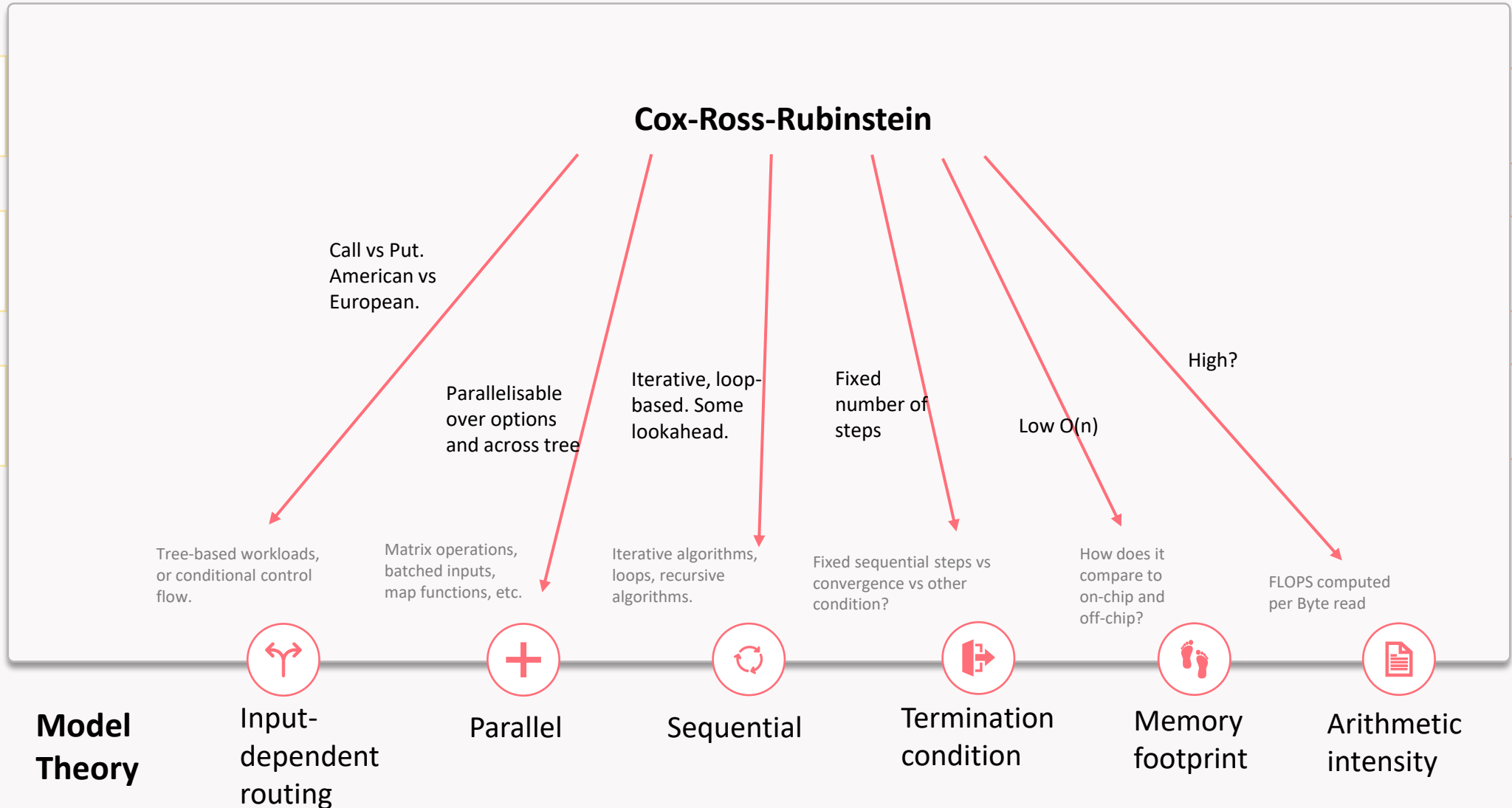
- Forward pass: create a tree of prices by assuming price of the option will increase/decrease by a specific factor
- Backward pass: iteratively calculate binomial value at the previous timepoint. This is the expected value discounted by the risk-free rate.



A MODEL MARRIAGE

Software
Implementation

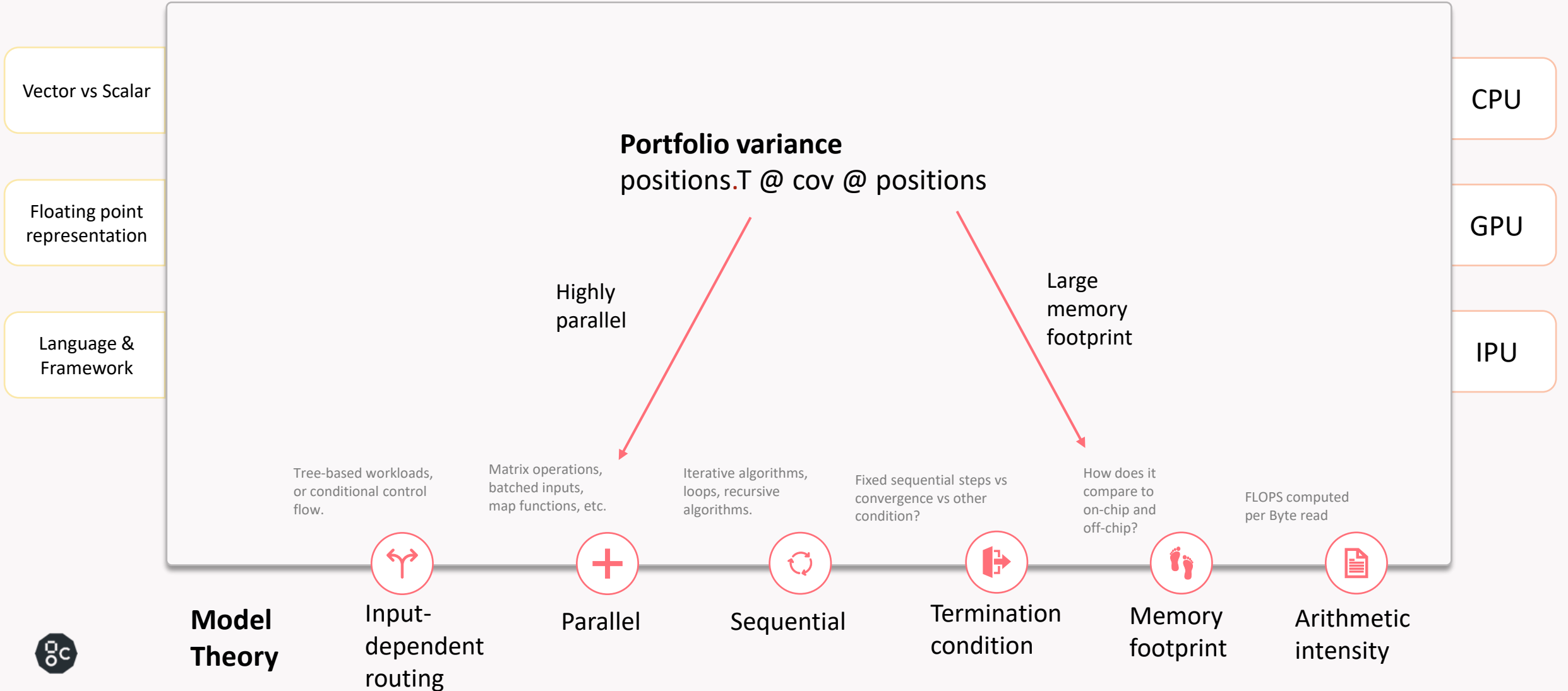
Hardware



A MODEL MARRIAGE

Software
Implementation

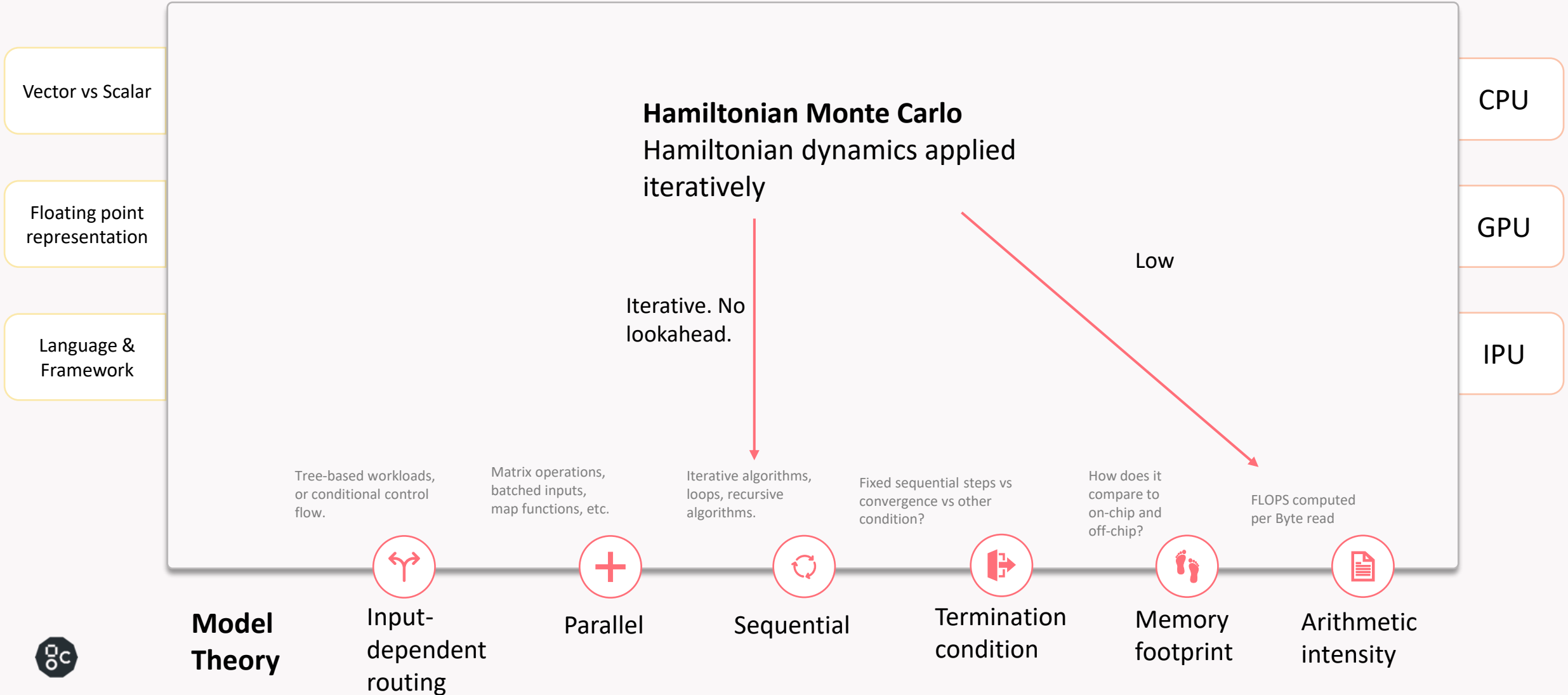
Hardware



A MODEL MARRIAGE

Software
Implementation

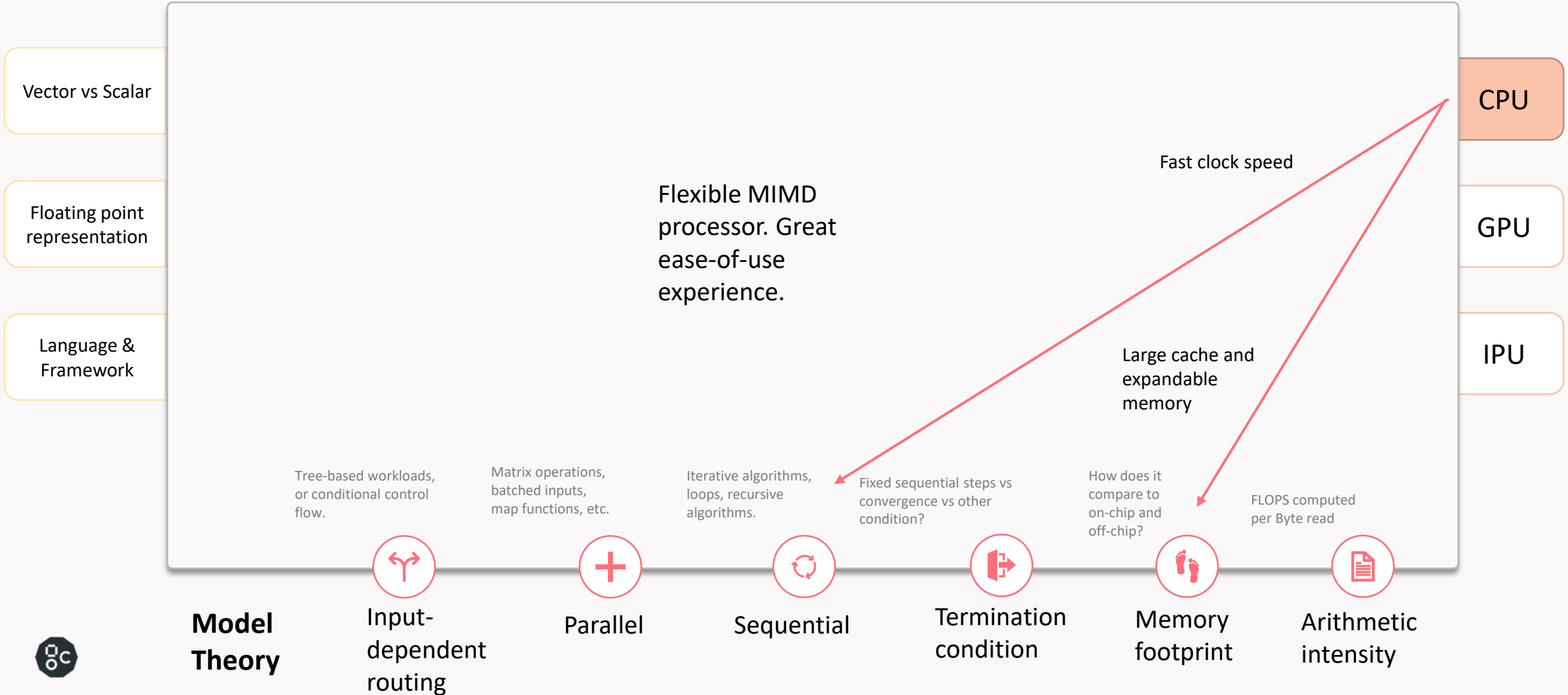
Hardware



A MODEL MARRIAGE

Software
Implementation

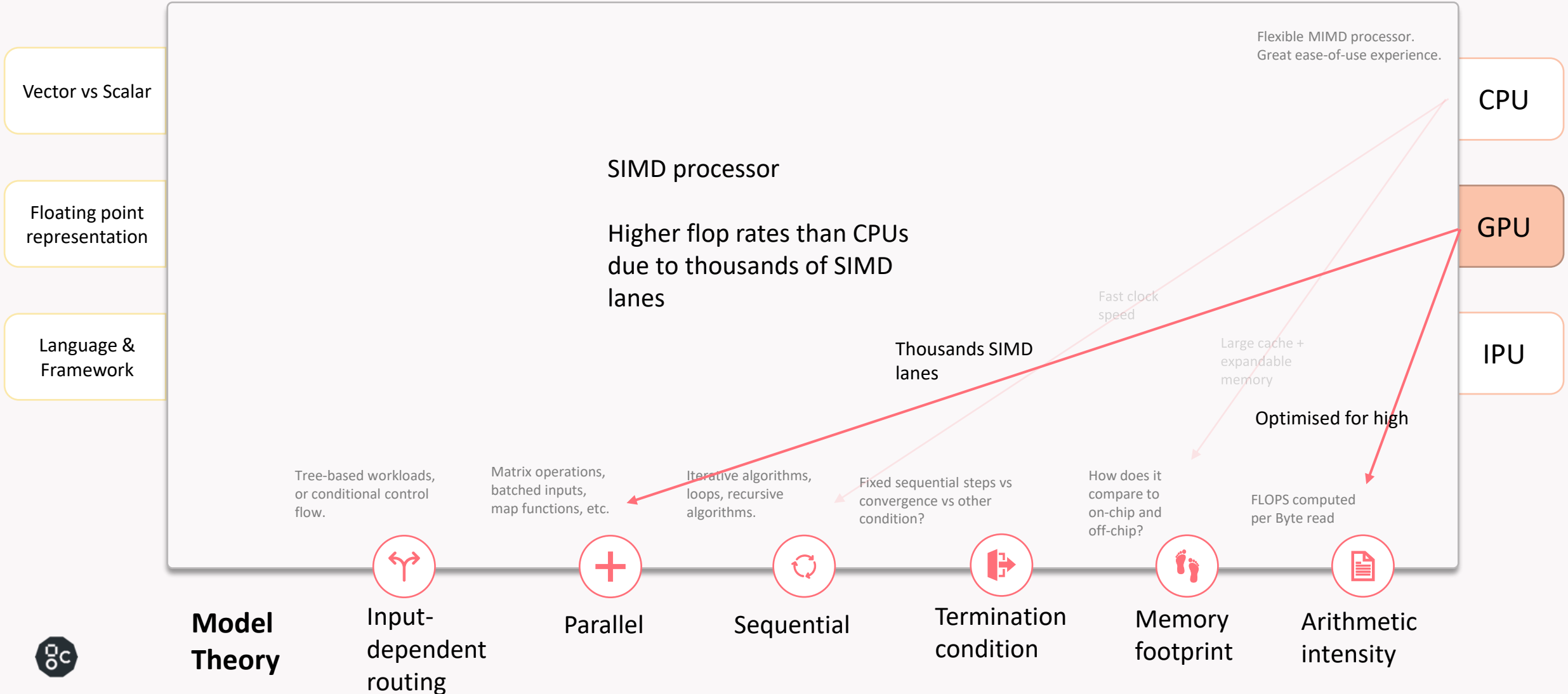
Hardware



A MODEL MARRIAGE

Software Implementation

Hardware



A MODEL MARRIAGE

Software Implementation

Hardware

Vector vs Scalar

Floating point representation

Language & Framework



CPU

GPU

IPU

MIMD processor, like a CPU.

1472 cores. FLOP rates comparable to GPUs.

900MB on-chip SRAM.

Tree-based workloads, or conditional control flow.

Matrix operations, batched inputs, map functions, etc.

Iterative algorithms, loops, recursive algorithms.

Fixed sequential steps vs convergence vs other condition?

How does it compare to on-chip and off-chip?

FLOPS computed per Byte read

Model Theory



Input-dependent routing



Parallel



Sequential



Termination condition



Memory footprint



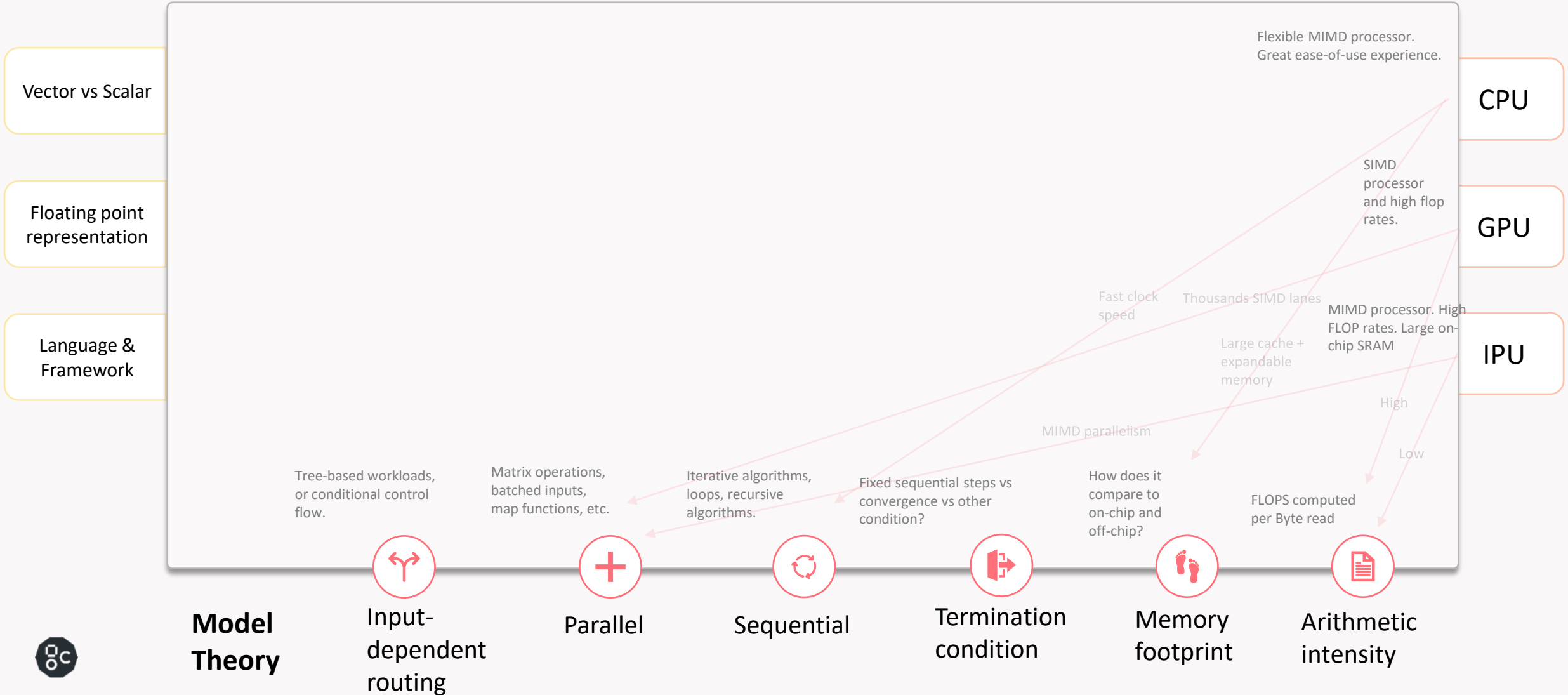
Arithmetic intensity



A MODEL MARRIAGE

Software
Implementation

Hardware



A MODEL MARRIAGE

Software
Implementation

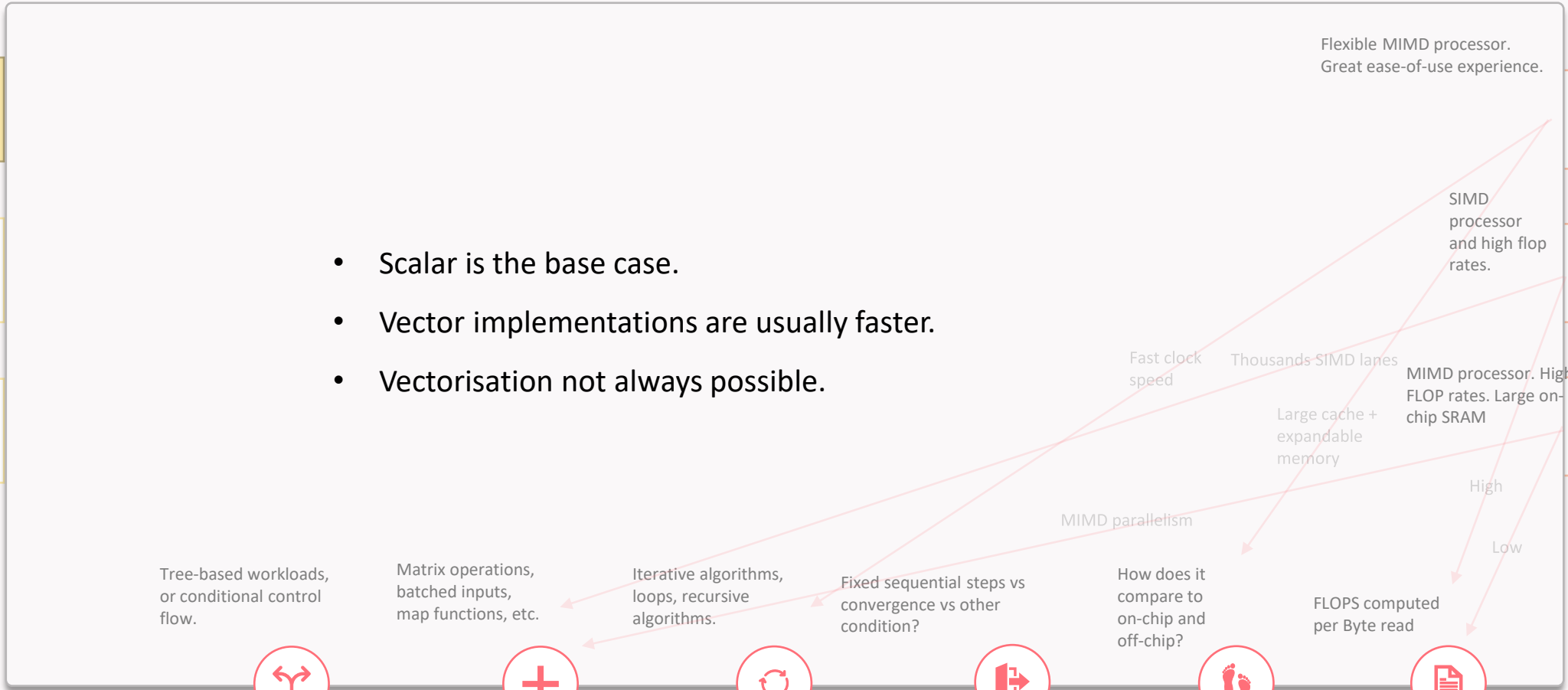
Hardware

Vector vs Scalar

Floating point
representation

Language &
Framework

- Scalar is the base case.
- Vector implementations are usually faster.
- Vectorisation not always possible.



CPU

GPU

IPU

Tree-based workloads,
or conditional control
flow.

Matrix operations,
batched inputs,
map functions, etc.

Iterative algorithms,
loops, recursive
algorithms.

Fixed sequential steps vs
convergence vs other
condition?

How does it
compare to
on-chip and
off-chip?

FLOPS computed
per Byte read



Model
Theory

Input-
dependent
routing

Parallel

Sequential

Termination
condition

Memory
footprint

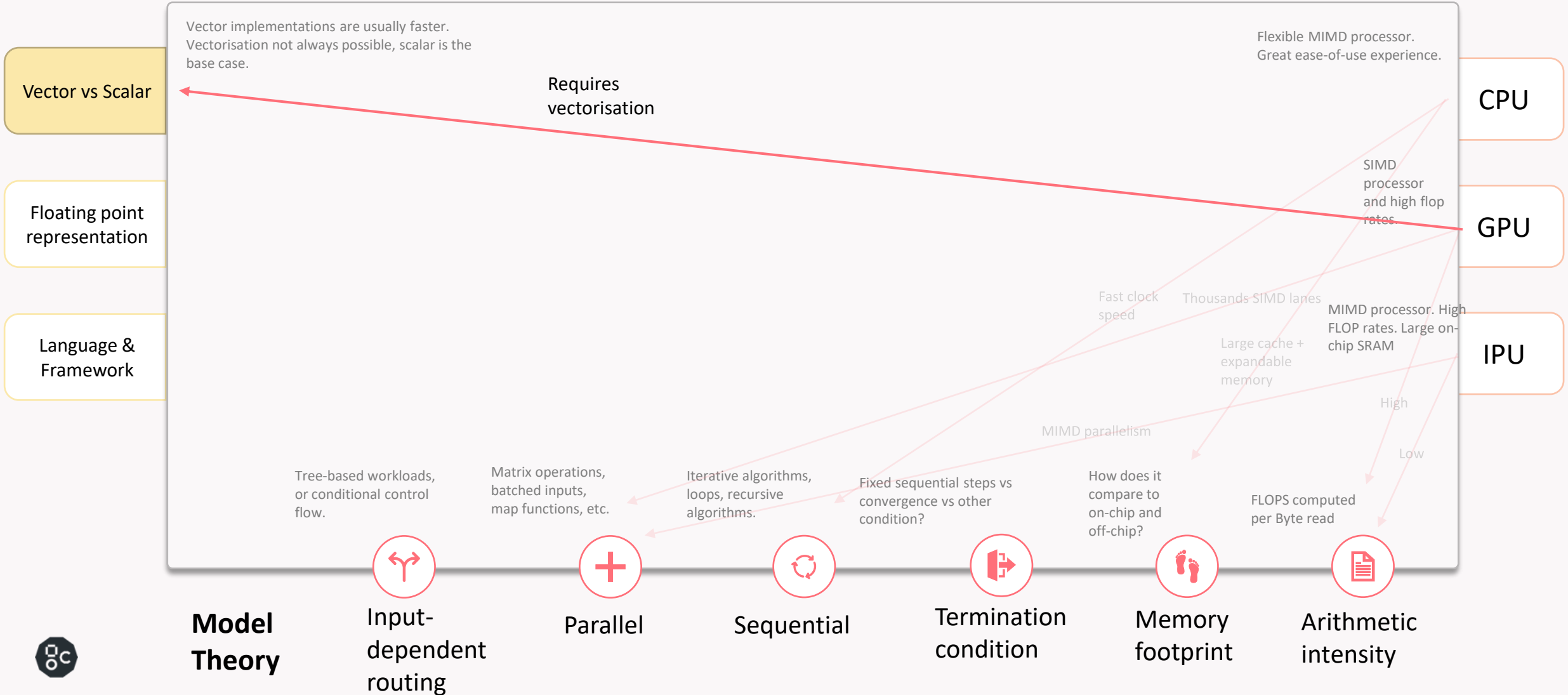
Arithmetic
intensity



A MODEL MARRIAGE

Software
Implementation

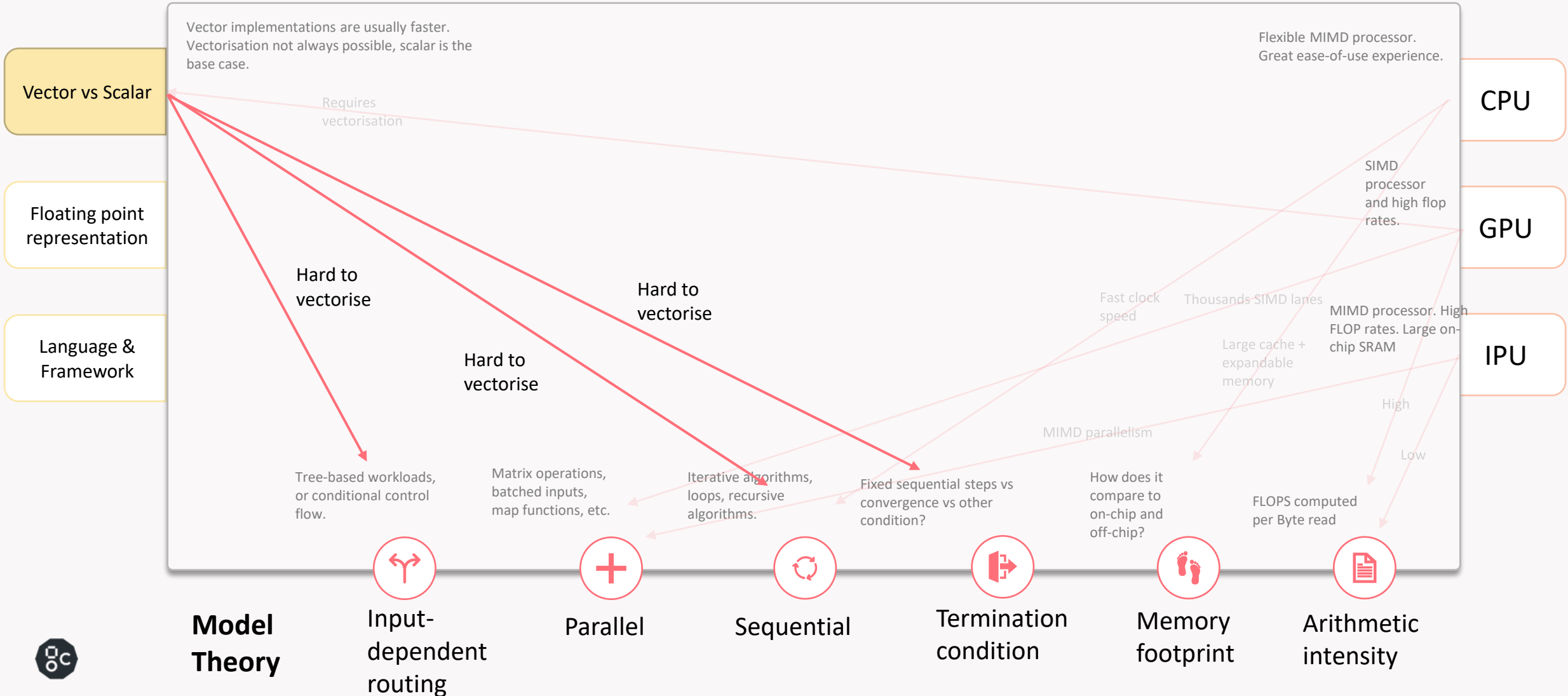
Hardware



A MODEL MARRIAGE

Software Implementation

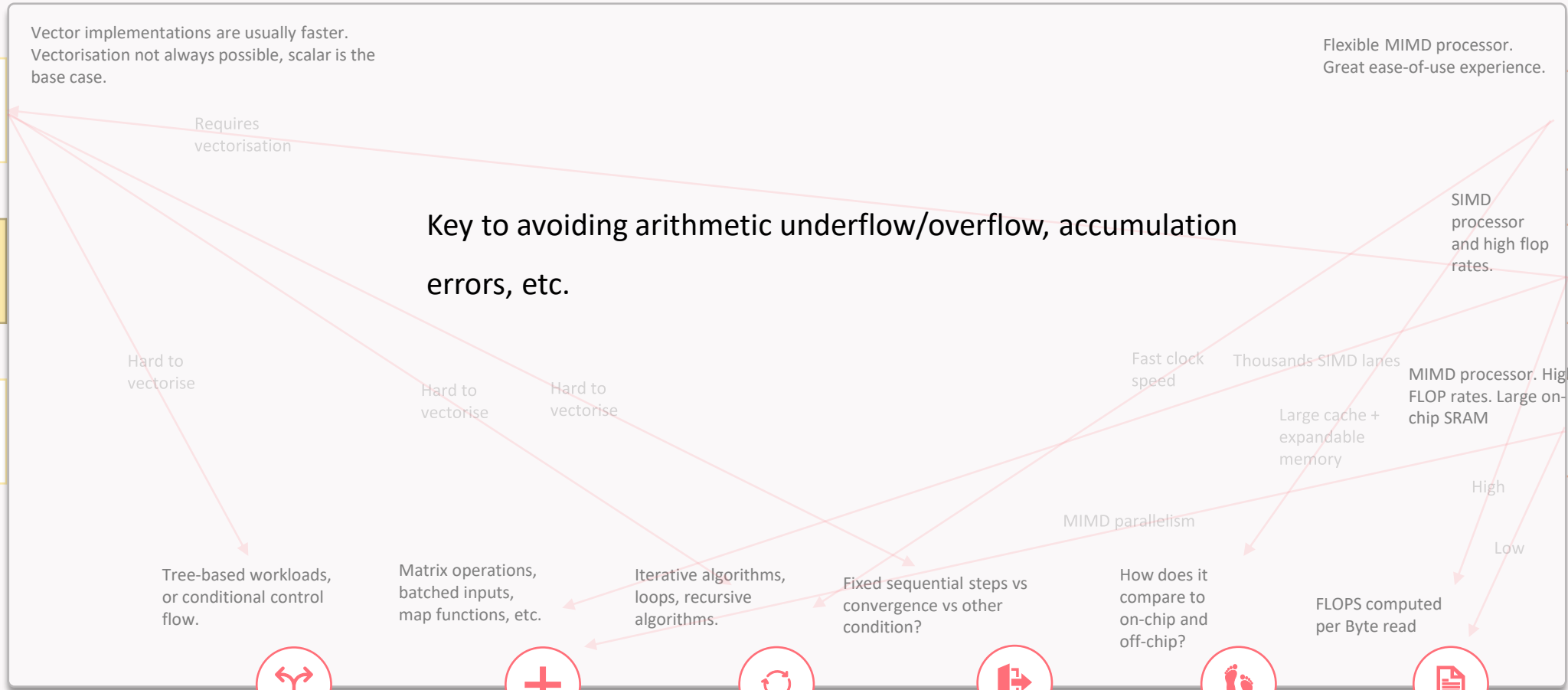
Hardware



A MODEL MARRIAGE

Software Implementation

Hardware



Model Theory



Input-dependent routing



Parallel



Sequential



Termination condition



Memory footprint



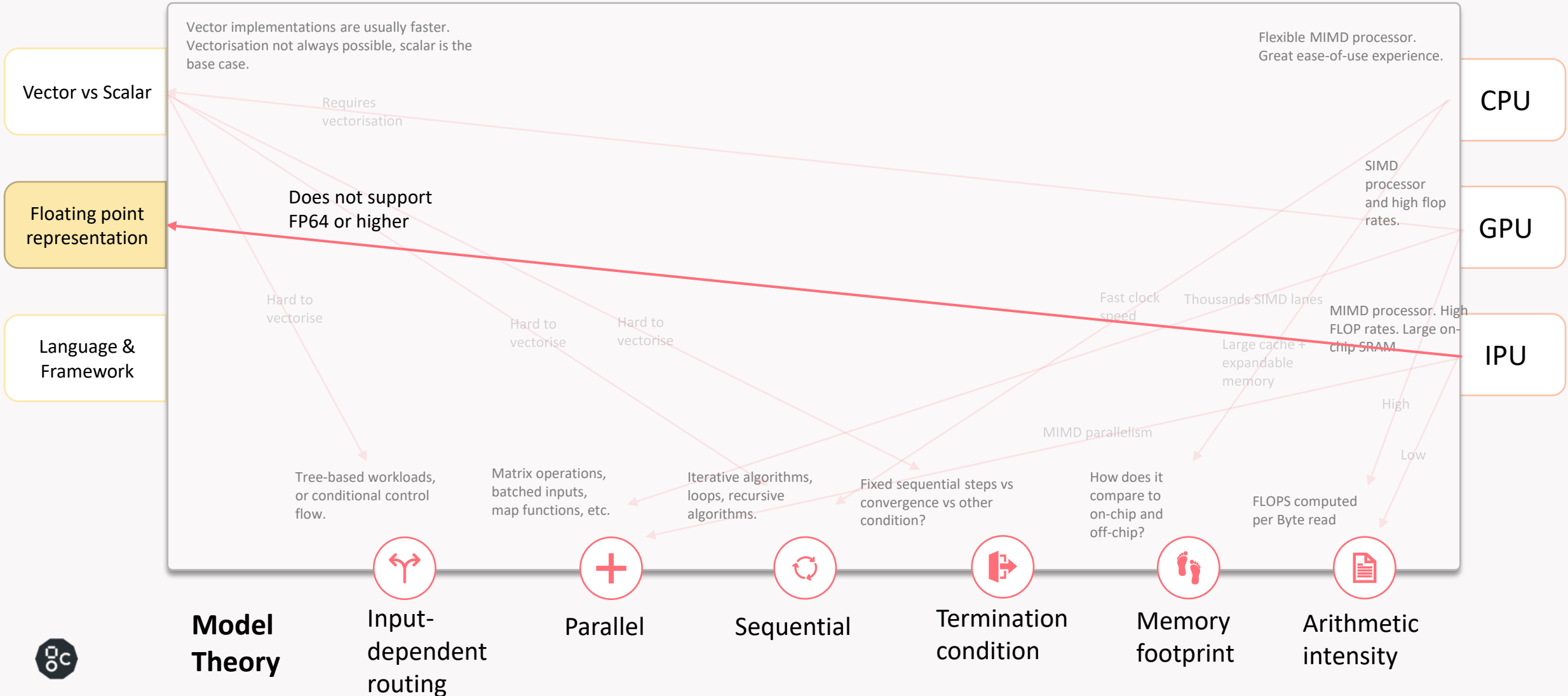
Arithmetic intensity



A MODEL MARRIAGE

Software Implementation

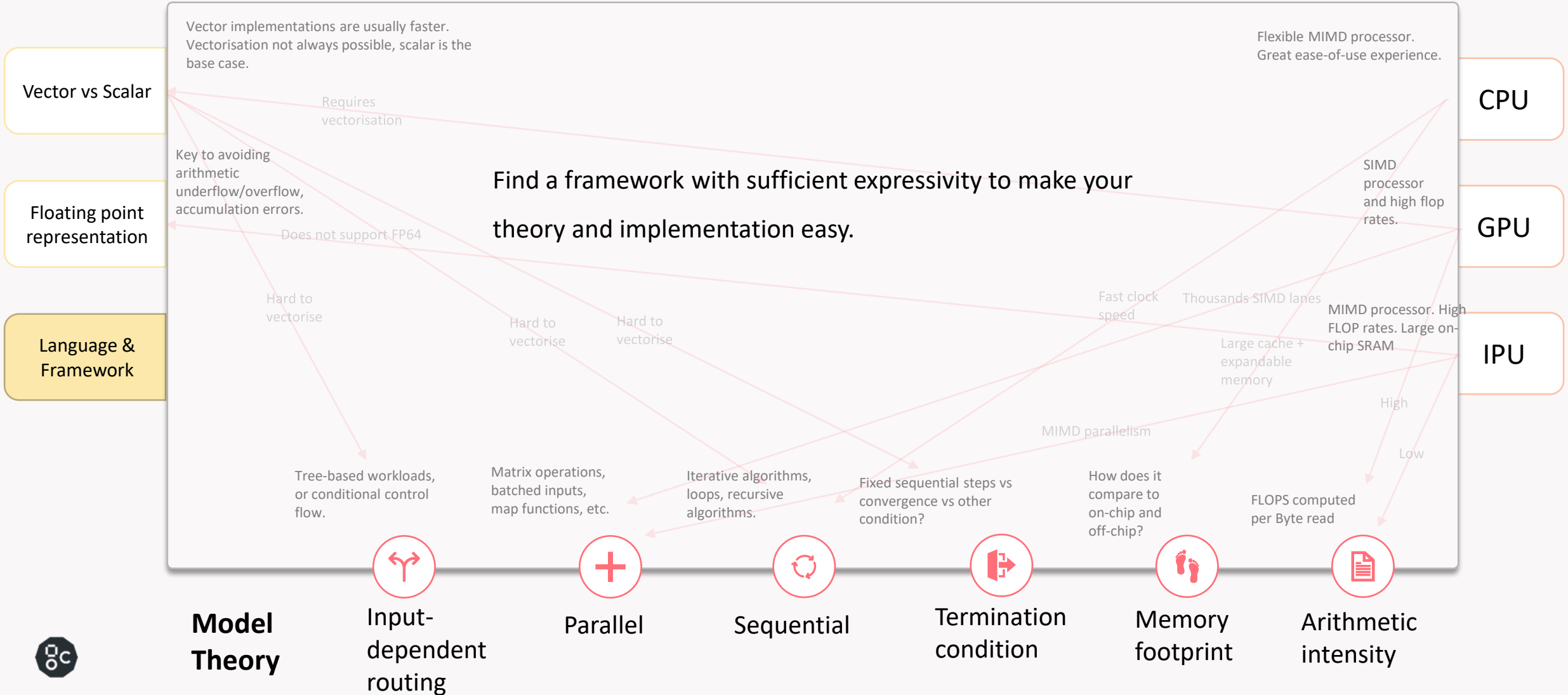
Hardware



A MODEL MARRIAGE

Software
Implementation

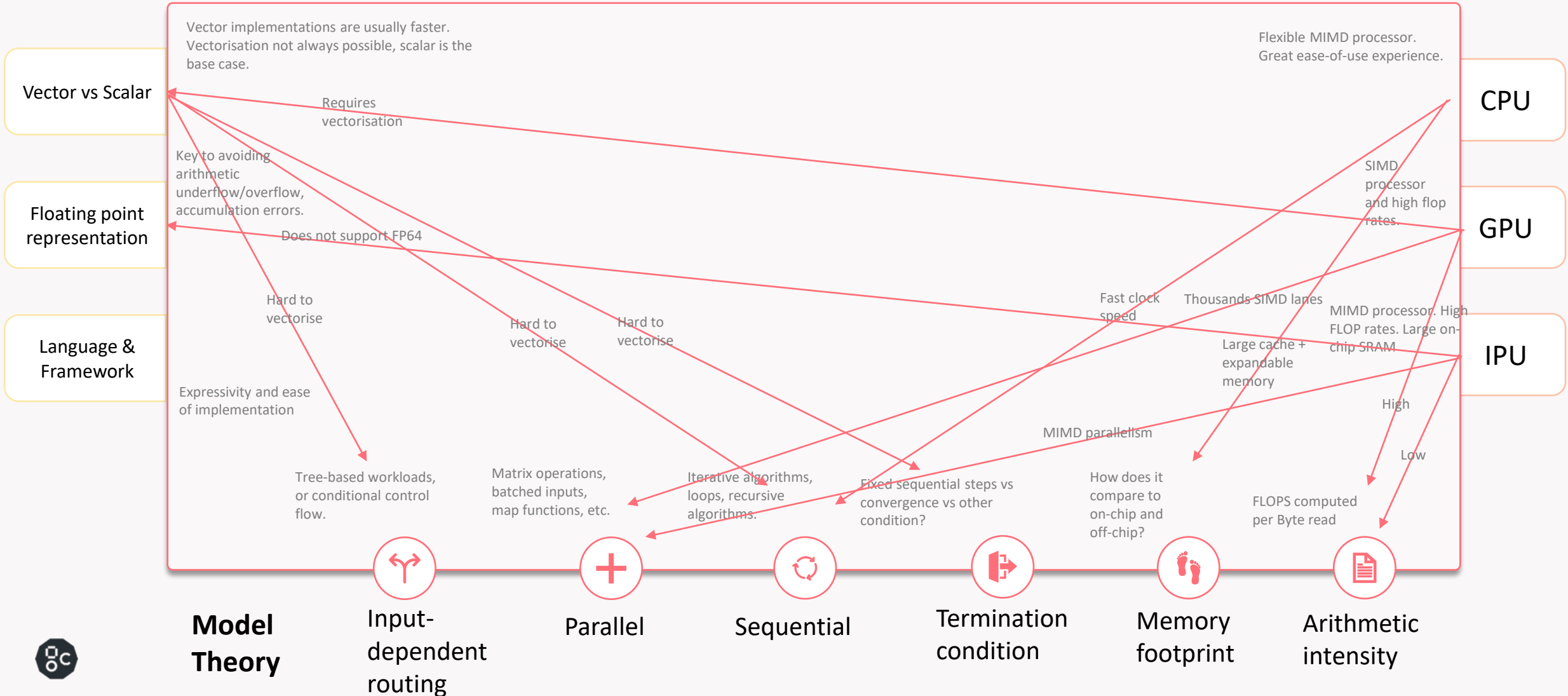
Hardware



A MODEL MARRIAGE

Software Implementation

Hardware



CASE STUDY – SOFTWARE AND HARDWARE



COX ROSS RUBINSTEIN

Software implementation

Scalar implementations are the base case.

Can be vectorized in multiple ways: across the tree, or across multiple options

```
double Binomial(int n, double Spot, double K, double r, double q, double v, double T, char PutCall, char OpStyle)
{
    int i, j;
    vector<vector<double>> S(n + 1, vector<double>(n + 1));
    vector<vector<double>> Op(n + 1, vector<double>(n + 1));
    double dt, u, d, p;

    dt = T / n;
    u = exp(v*sqrt(dt));
    d = 1 / u;
    p = (exp((r - q)*dt) - d) / (u - d);

    // Build the binomial tree
    for (j = 0; j <= n; j++) {
        for (i = 0; i <= j; i++) {
            S[i][j] = Spot*pow(u, j - i)*pow(d, i);
        }
    }

    // Compute terminal payoffs
    for (i = 0; i <= n; i++) {
        if (PutCall == 'C')
            Op[i][n] = max(S[i][n] - K, 0.0);
        else
            Op[i][n] = max(K - S[i][n], 0.0);
    }

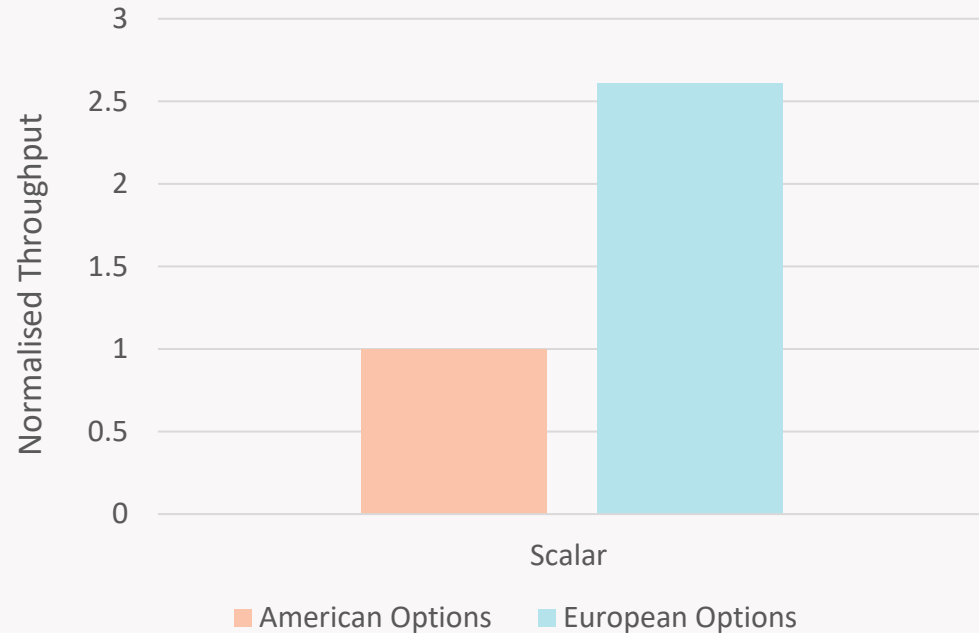
    // Backward recursion through the tree
    for (j = n - 1; j >= 0; j--) {
        for (i = 0; i <= j; i++) {
            if (OpStyle == 'E')
                Op[i][j] = exp(-r*dt)*(p*(Op[i][j + 1]) + (1 - p)*(Op[i + 1][j + 1]));
            else {
                if (PutCall == 'C')
                    Op[i][j] = max(S[i][j] - K, exp(-r*dt)*(p*(Op[i][j + 1]) + (1 - p)*(Op[i + 1][j + 1])));
                else
                    Op[i][j] = max(K - S[i][j], exp(-r*dt)*(p*(Op[i][j + 1]) + (1 - p)*(Op[i + 1][j + 1])));
            }
        }
    }

    return Op[0][0];
}
```

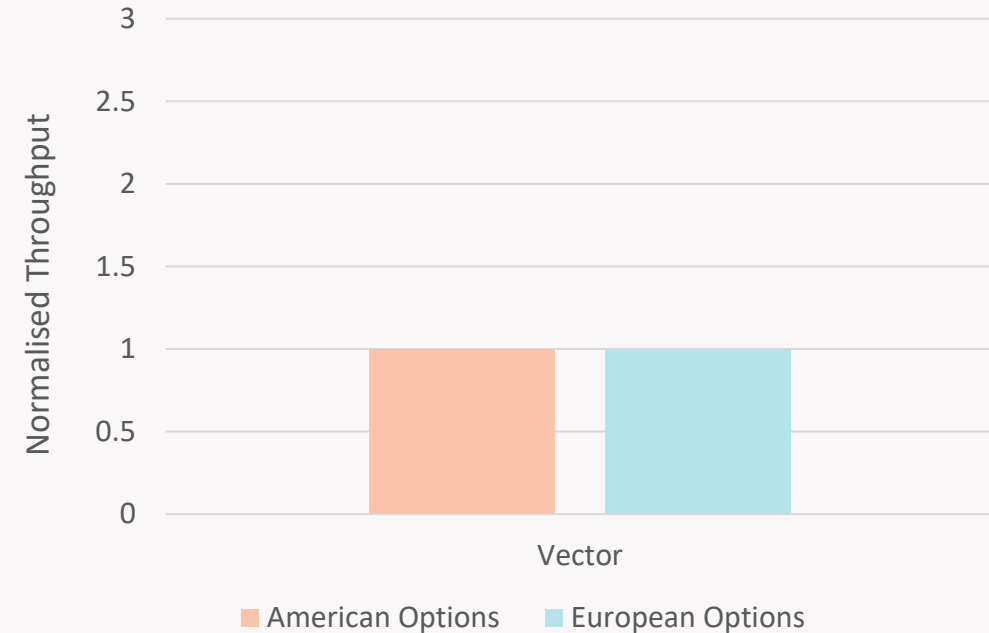


VECTORISED IMPLEMENTATIONS CAN HAVE SIDE EFFECTS

Normalised Throughput – **Scalar Code**
American vs European Options



Normalised Throughput – **Vectorised Code**
American vs European Options



AUTOCALLABLES

Break statements inside for loops...
vectorization=hard.

```
def AutoCallableNote(valuationDate, couponDates, strike, pastFixings,
                    autoCallBarrier, couponBarrier, protectionBarrier, hasMemory, finalRedemptionFormula,
                    coupon, notional, dayCounter, process, generator, nPaths, curve):

    # immediate exit trigger for matured transaction
    if(valuationDate >= couponDates[-1]): return 0.0

    # immediate exit trigger for any past autocall event
    if(valuationDate >= couponDates[0]):
        if(max(pastFixings.values()) >= (autoCallBarrier * strike)): return 0.0

    # create date array for path generator
    # combine valuation date and all the remaining coupon dates
    dates = np.hstack((np.array([valuationDate]), couponDates[couponDates > valuationDate]))

    # generate paths for a given set of dates, exclude the current spot rate
    paths = generator(dates, dayCounter, process, nPaths)[:,:1:]

    # identify the past coupon dates
    pastDates = couponDates[couponDates <= valuationDate]

    # conditionally, merge given past fixings from a given dictionary and generated paths
    if(pastDates.shape[0] > 0):
        pastFixingsArray = np.array([pastFixings[pastDate] for pastDate in pastDates])
        pastFixingsArray = np.tile(pastFixingsArray, (paths.shape[0], 1))
        paths = np.hstack((pastFixingsArray, paths))

    # result accumulator
    global_pv = []
    expirationDate = couponDates[-1]
    hasMemory = int(hasMemory)

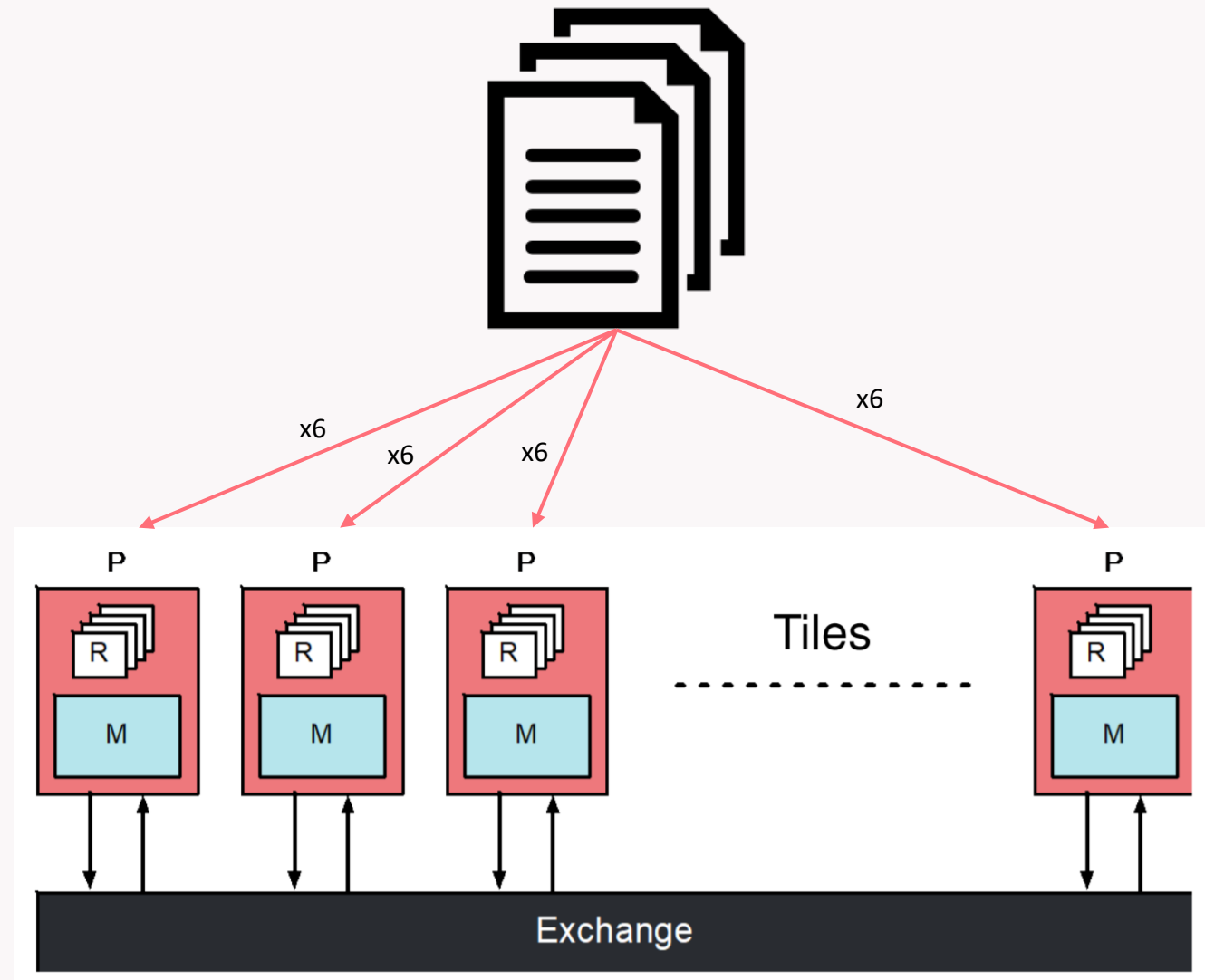
    # loop through all simulated paths
    for path in paths:
        payoffPV = 0.0
        unpaidCoupons = 0
        hasAutoCalled = False

        # loop through set of coupon dates and index ratios
        for date, index in zip(couponDates, (path / strike)):
            # if autocall event has been triggered, immediate exit from this path
            if(hasAutoCalled): break
            payoff = 0.0

        # payoff calculation at expiration
        if(date == expirationDate):
            # index is greater or equal to coupon barrier
```



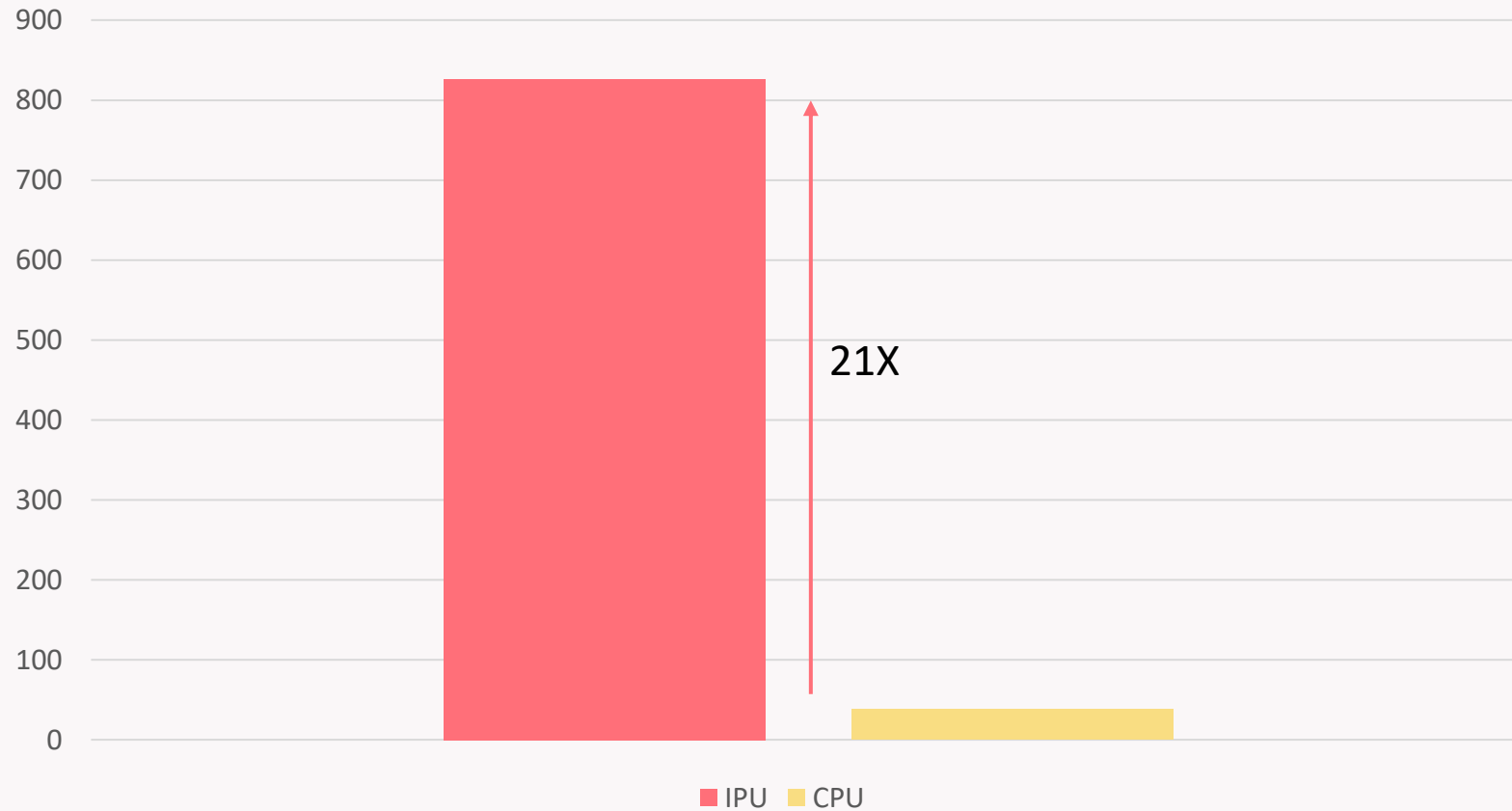
MAPPING OPTIONS TO TILES



OPTION PRICING BENCHMARK

Throughput comparison between IPU and CPU

American Call Options – IPU vs CPU



Why is the IPU faster?

- 50x more cores
- 20x the on-chip memory
- The IPU maintains enough on-chip memory to process an option with each of its 1472x6 threads

Not a STAC benchmark



THANKYOU

For:

- Updates on our GPU benchmarking
- Updates on IPU performance
- Trying out Cox-Ross-Rubinstein out on IPU for yourself!
- Any other Graphcore related queries

Email lukem@graphcore.ai



APPENDIX



ABSTRACT

"Model" marriages: Pairing implementations and hardware".

Box famously said that all models are wrong but some are useful. Financial firms could add a proviso: “a model is only useful if we can execute it quickly and efficiently”. High performance depends on three variables: the problem size/shape, the implementation code, and the hardware platform. But financial models vary widely, hardware choices expand every few months, and implementation options abound.

Technologists can benefit significantly from a systematic approach to understanding how the three variables interact. Luke has just such an approach. With the help of a real-life study that Graphcore performed for a hedge fund (optimizing Cox-Ross-Rubenstein to price millions of options), Luke will explain a process that categorizes and grades workloads and determines the best pairing of implementation and hardware.