Optimizing Dynamic Graph Processing on Multicores with the Locality-First Strategy

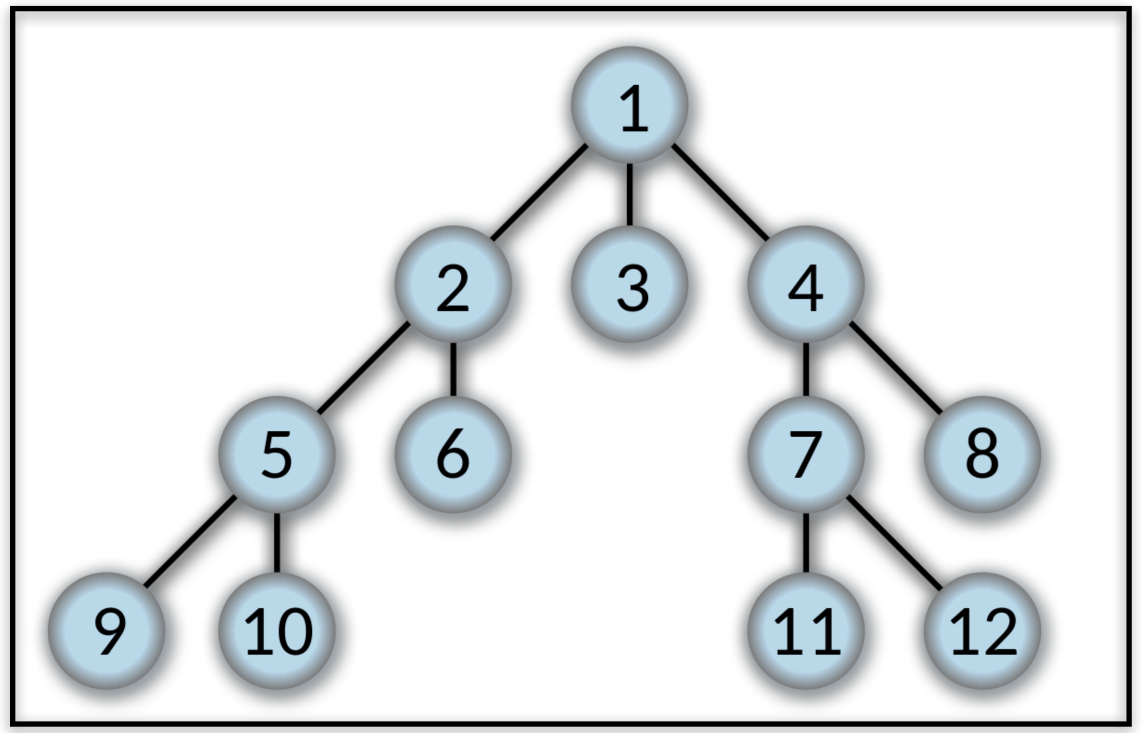
Helen Xu
MIT / Lawrence Berkeley National Laboratory
https://people.csail.mit.edu/hjxu

Example Problem: Fast Multicore Graph Processing

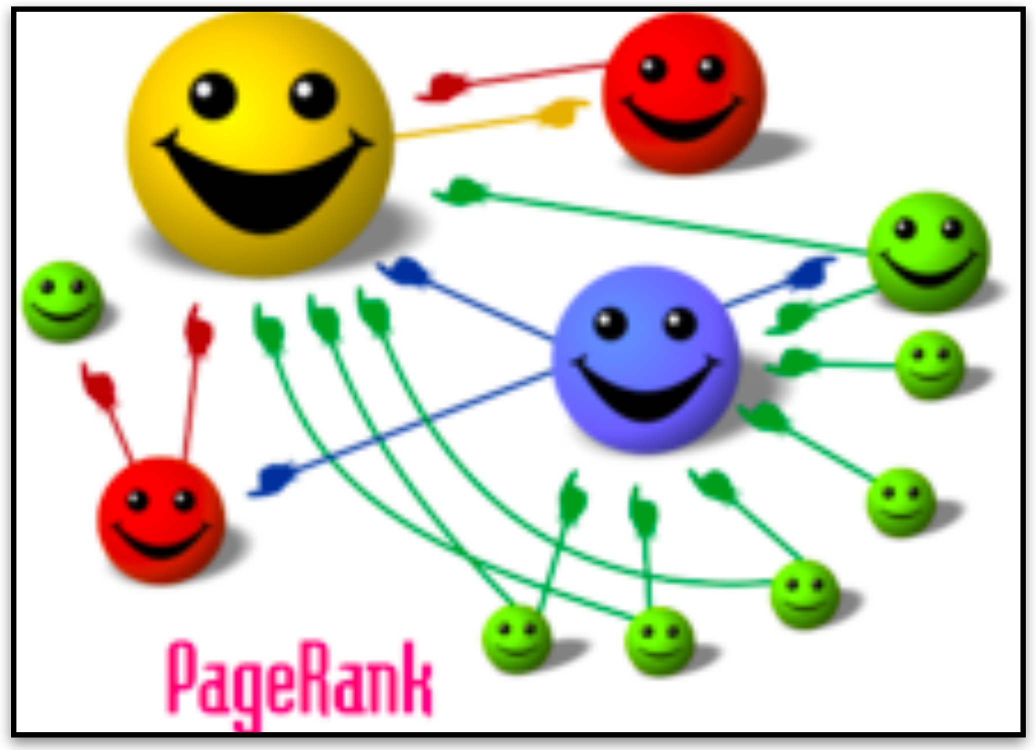
One example of a multicore algorithm optimization problem is **graph processing** [EdigerMcRiBa12, KyrolaBlGu12, ShunBl13, MackoMaMaSe15, DhulipalaBlSh19, BusatoGrBoBa18, GreenBa16].

Many large graph datasets (e.g. the Twitter graph) can fit into the **primary memory** of a single multicore.

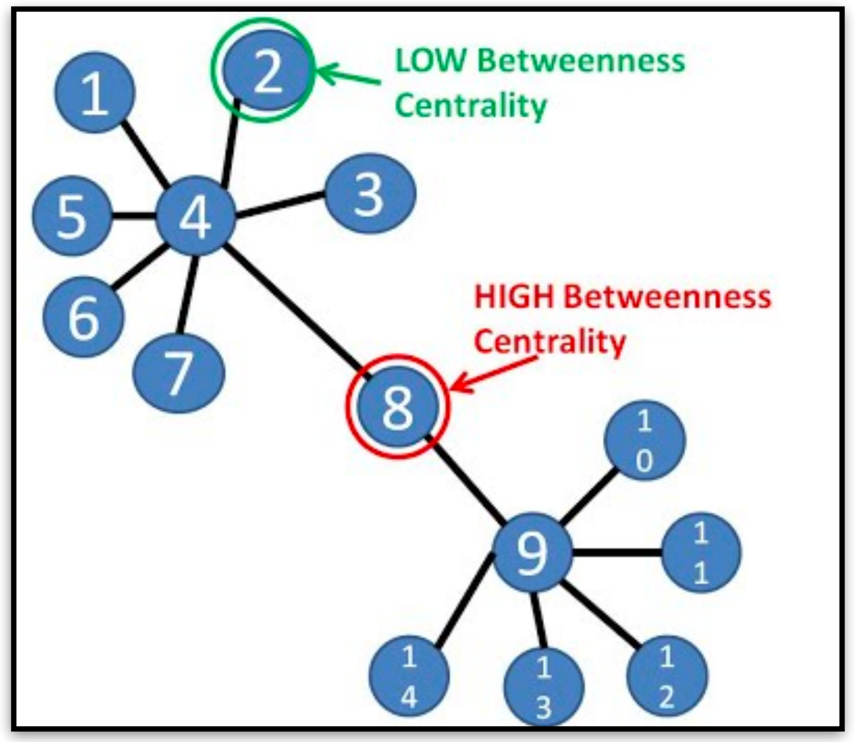
Example graph **queries**, or algorithms:



Breadth-first search



PageRank



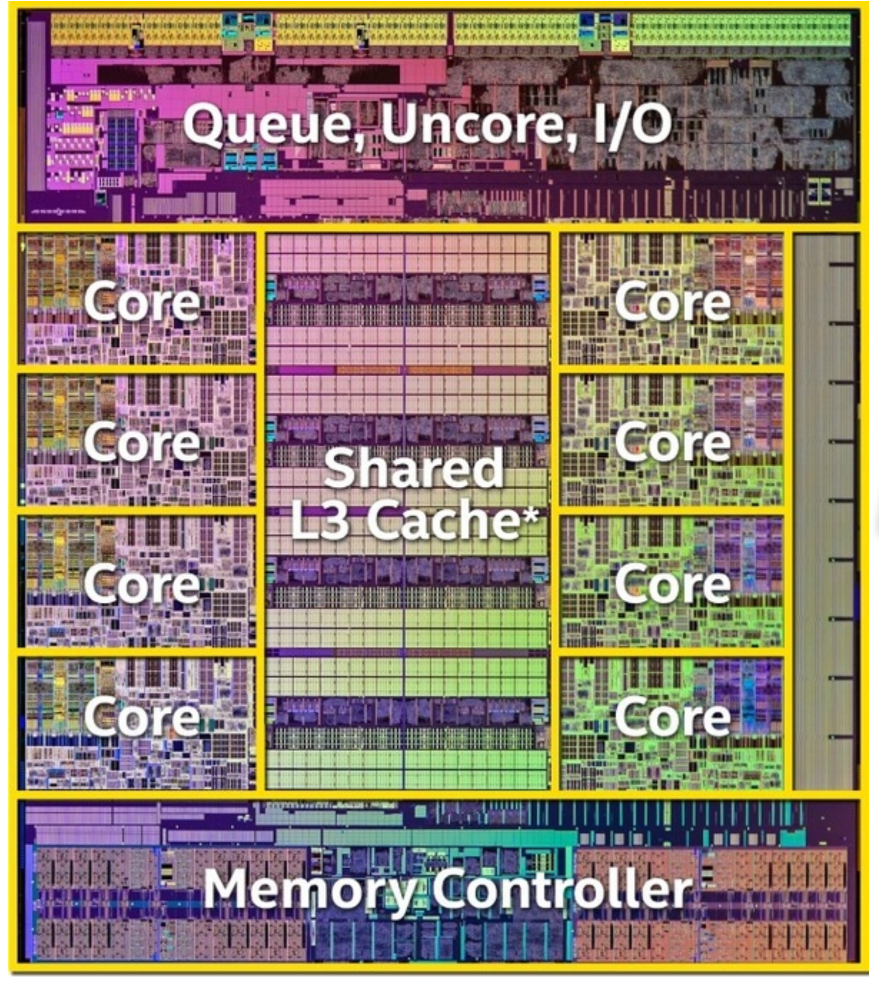
Betweenness centrality

Goal: Make these queries run as fast as possible.

Optimizing Programs on Multicores

Today's multicores are **widely accessible** to general programmers and **relatively cheap** compared to special-purpose hardware.

Two salient features of multicores are 1) **multiple cores** and 2) a steep (multilevel) **cache hierarchy**.

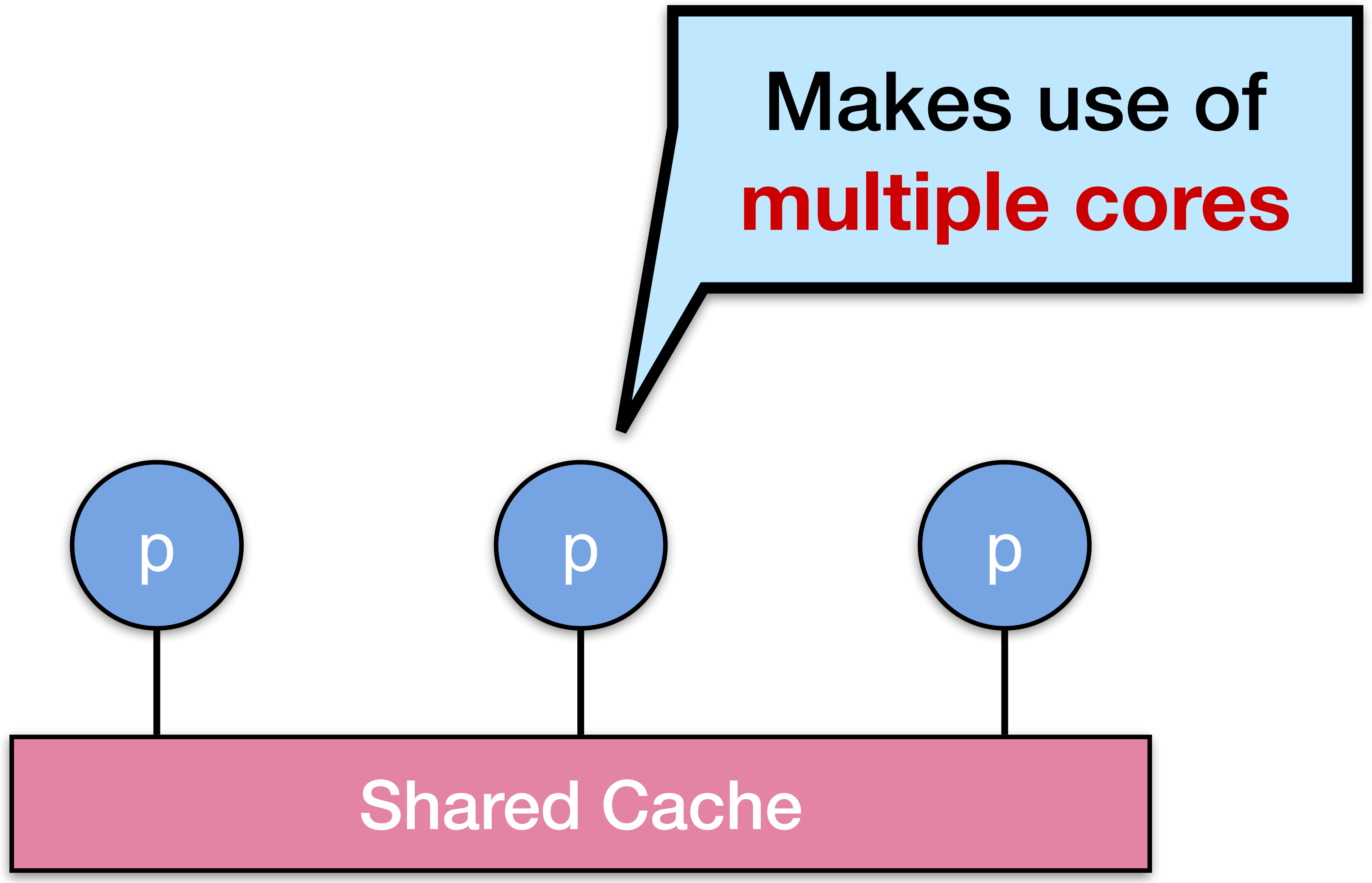


E.g. Intel Haswell

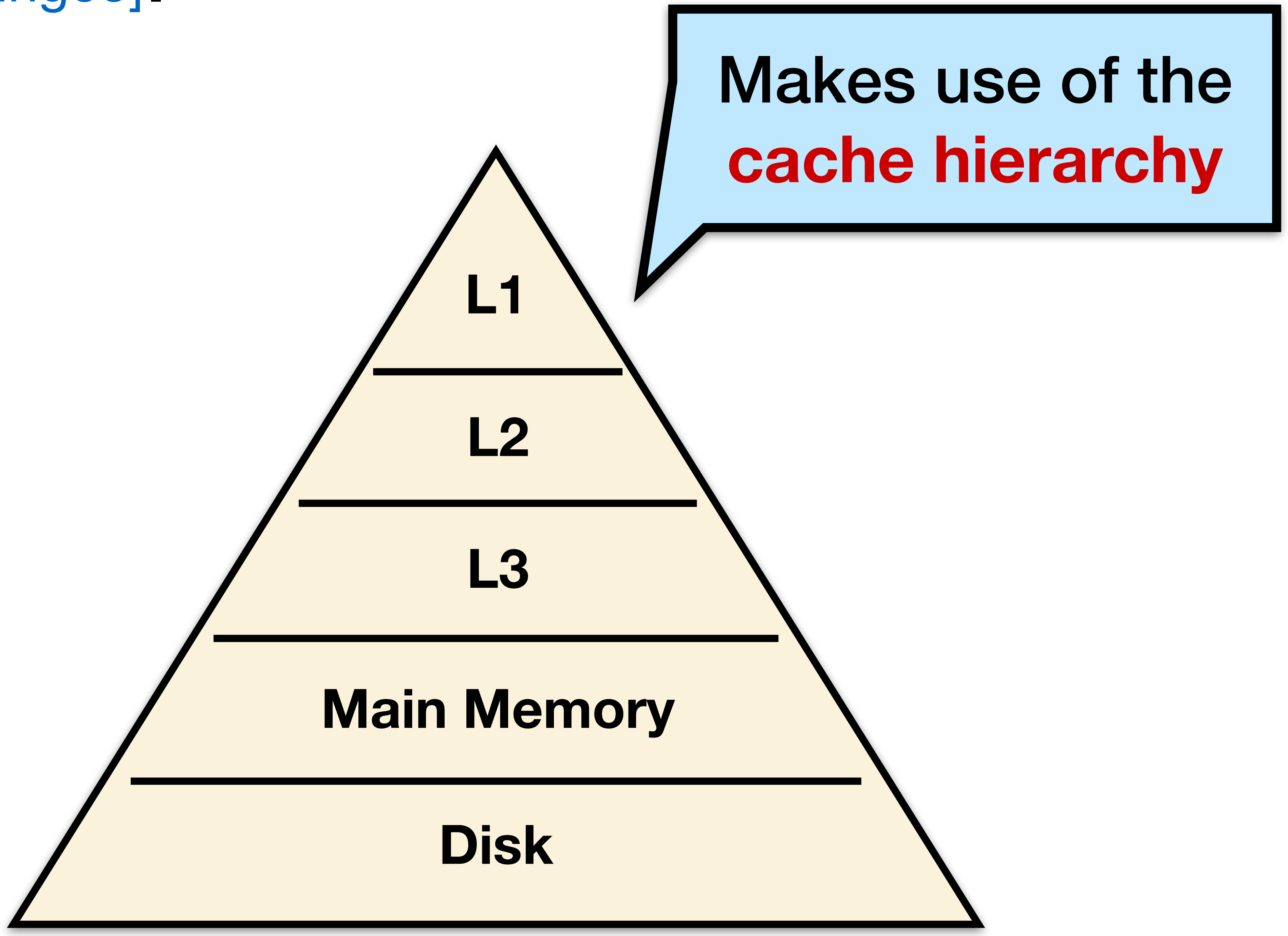
Writing fast code on these platforms is **notoriously hard** because of these features [Shun15, Schardl16, Kaler20, and many others]

Adapting to Multicore Hardware Features

Parallelism: the ability to perform multiple operations at the same time [Flynn72].



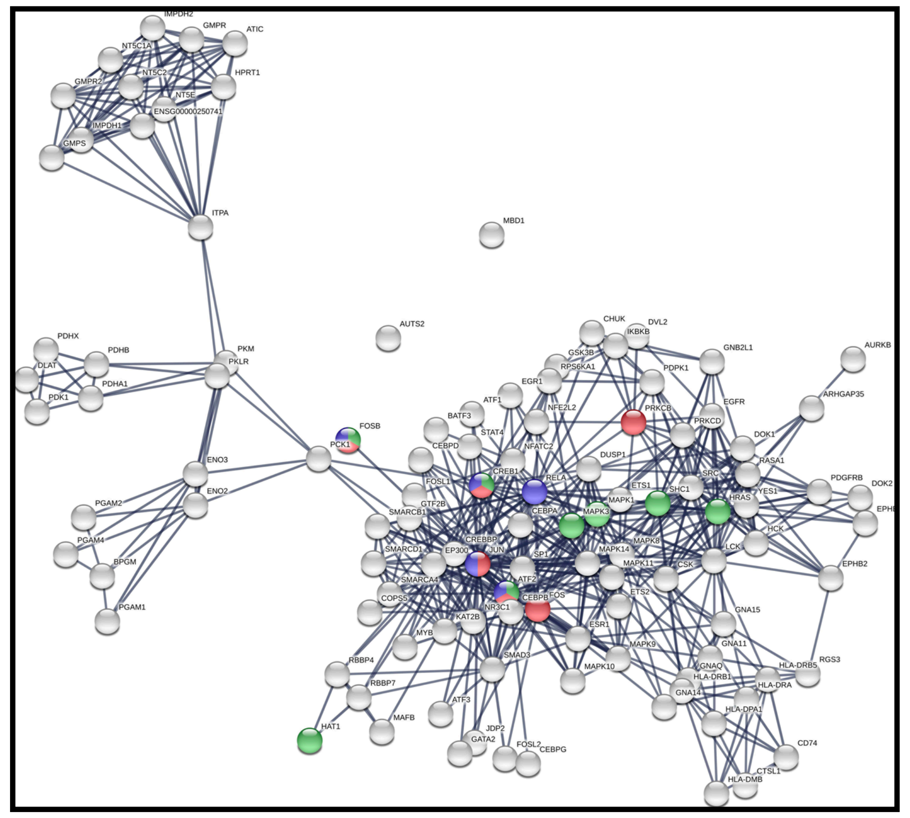
Locality: the tendency of programs to access the same or similar data over time [Denning72, Denning05].



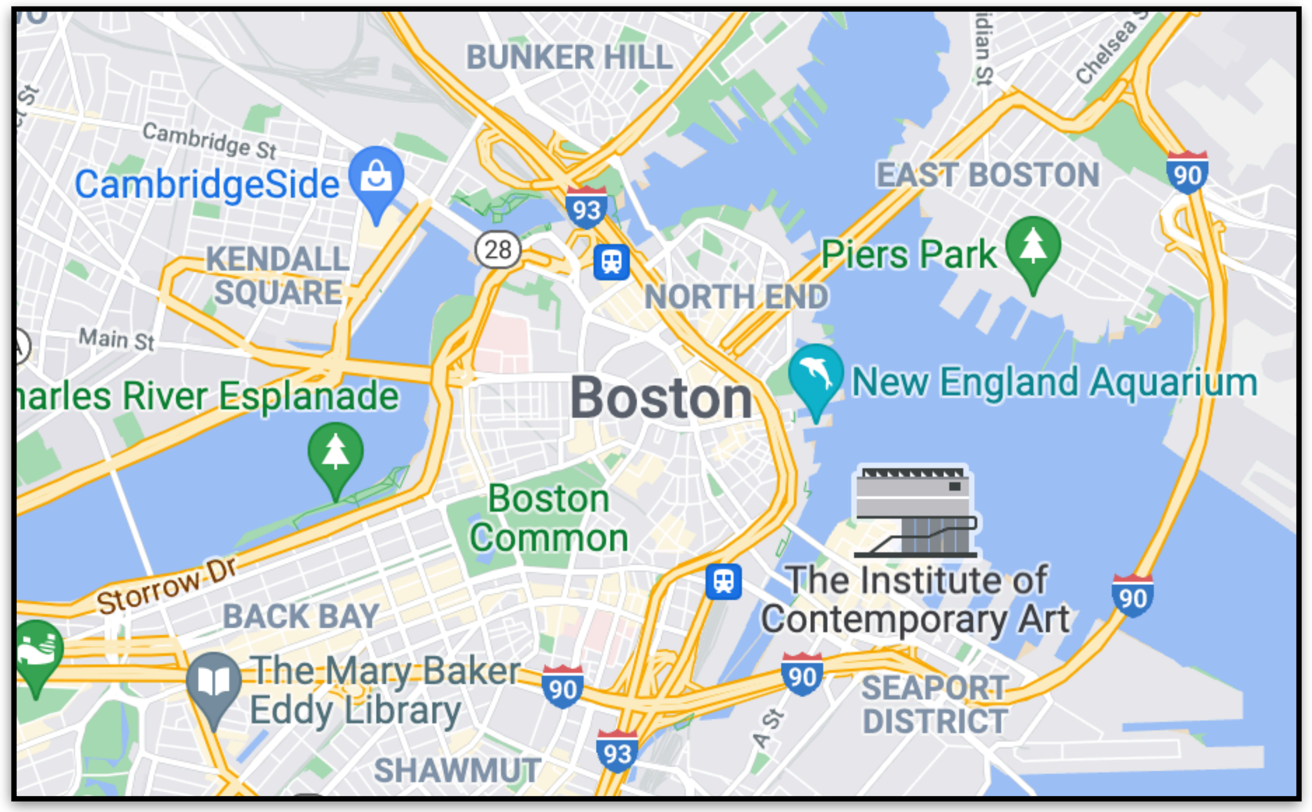
Real-World Graphs Are Sparse

Sparse graphs, which have many fewer edges than the total possible number of edges, underlie most real-world applications.

Social networks



Computational biology



Road networks

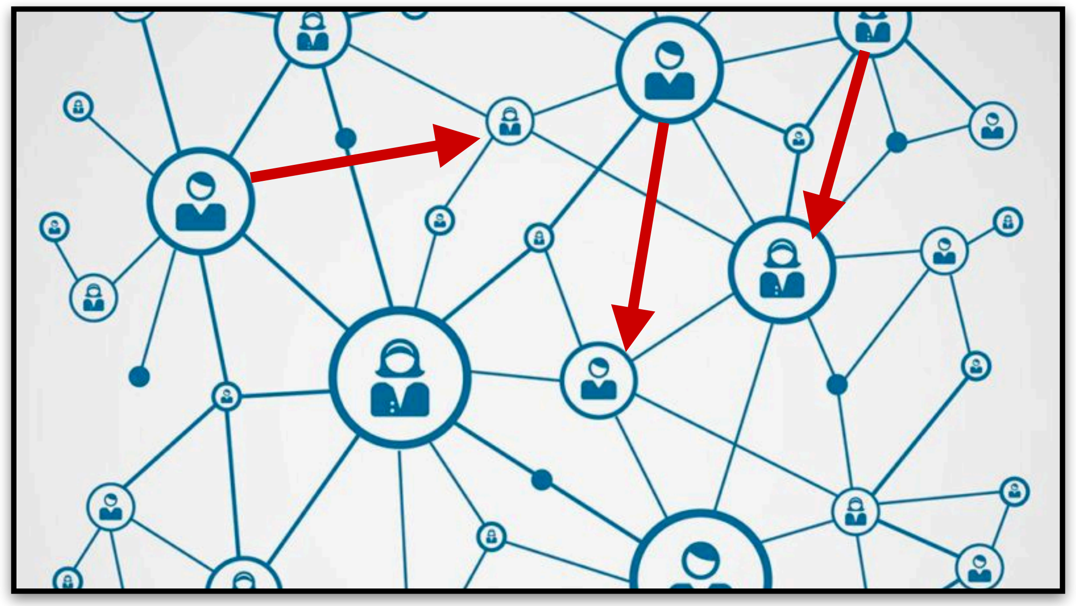
...and others!

Sparsity **disrupts locality** due to the presence of many zeroes in the data.

Real-World Graphs Are Also Dynamic

Furthermore, many real-world sparse graphs are **dynamic**: they **change over time**.

Updates



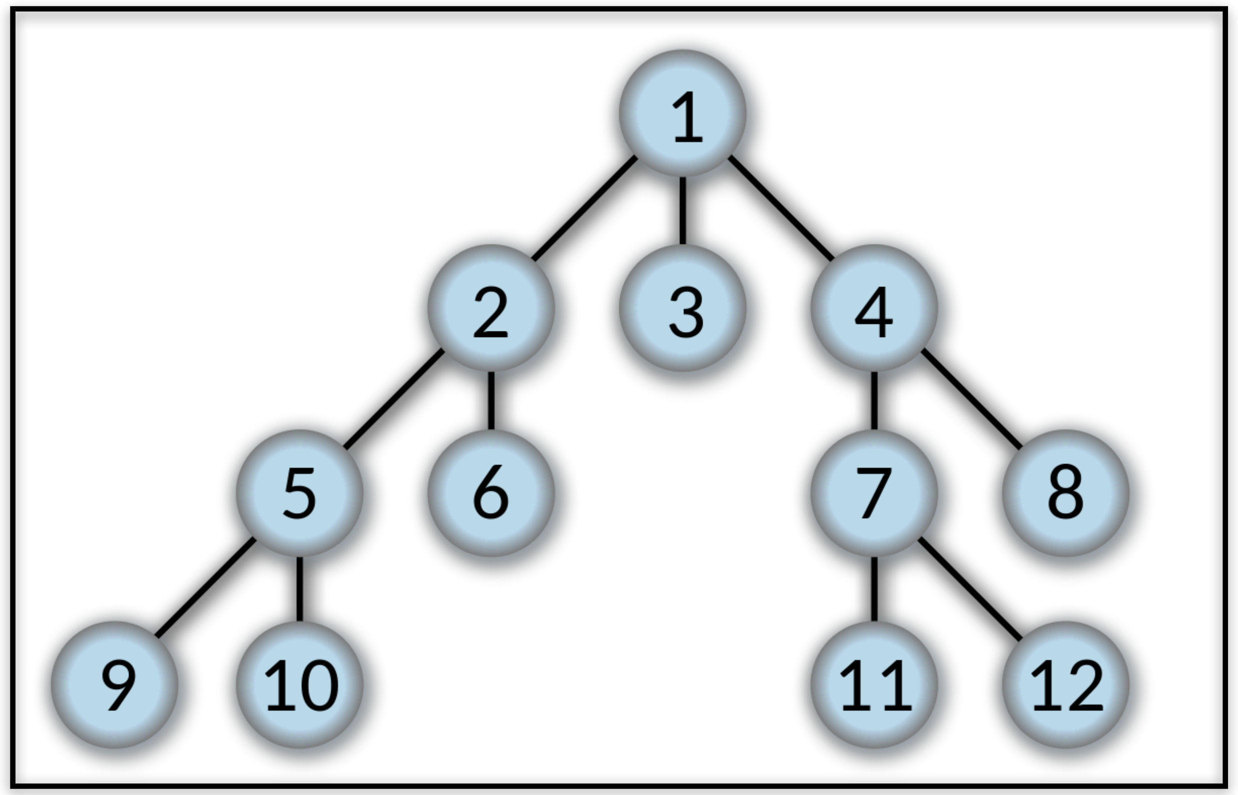
Systems for processing dynamic graphs support **updates** (e.g. edge insertions and deletions) and **queries** (algorithms run on the graph).

Dynamic graphs disrupt locality because of the inherent **tradeoff between colocating and updating** data.

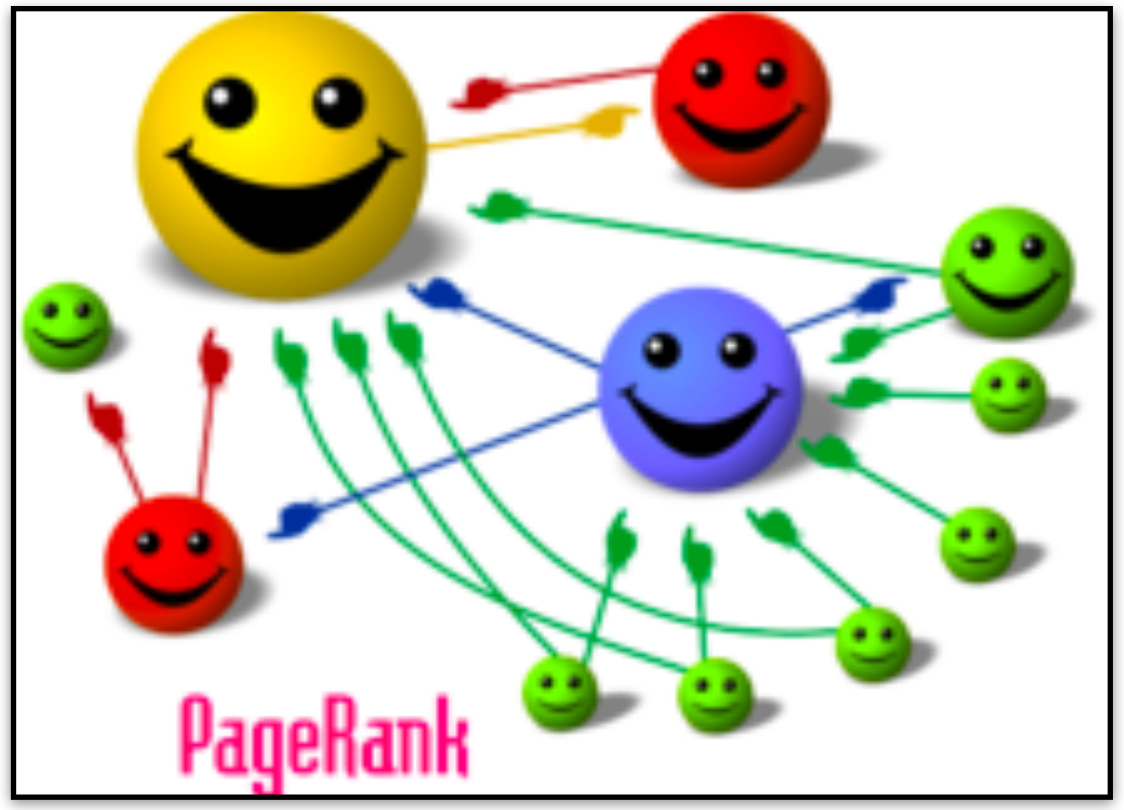
Multicore Optimization Enables Fast Graph Queries and Updates

Despite these challenges to locality, high-performance dynamic-graph-processing systems such as Aspen [DhulipalaShBl19] have taken huge steps towards **efficient queries and updates**.

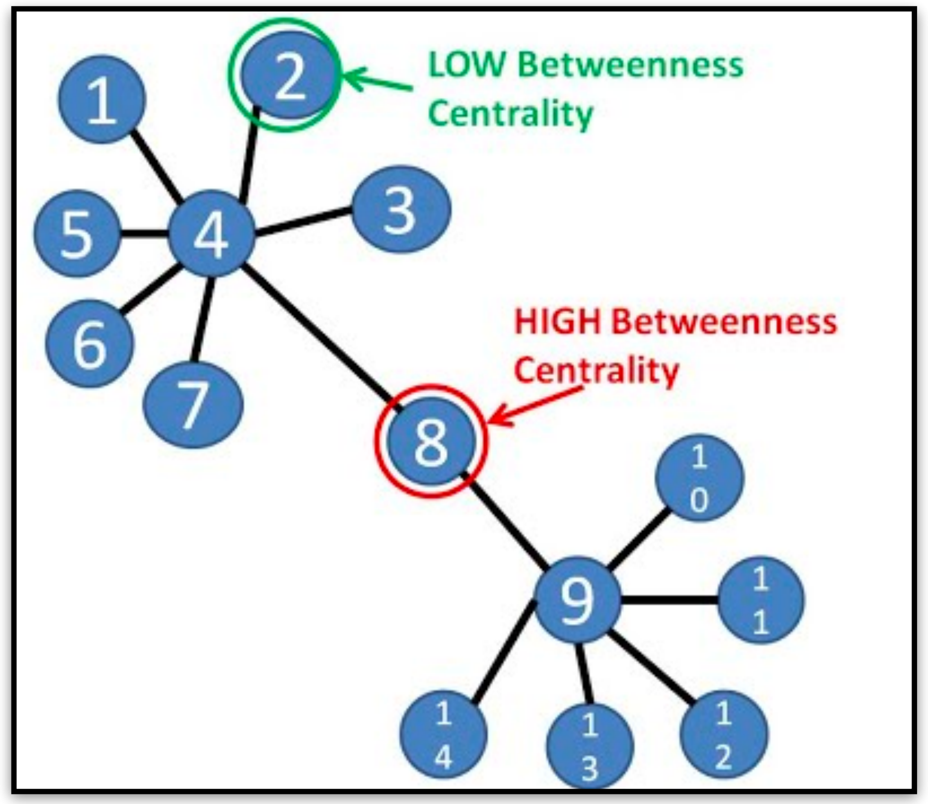
On 48 cores, Aspen runs the following queries on **Twitter** (2.4B edges):



Breadth-first search
0.32s



PageRank
24.03s



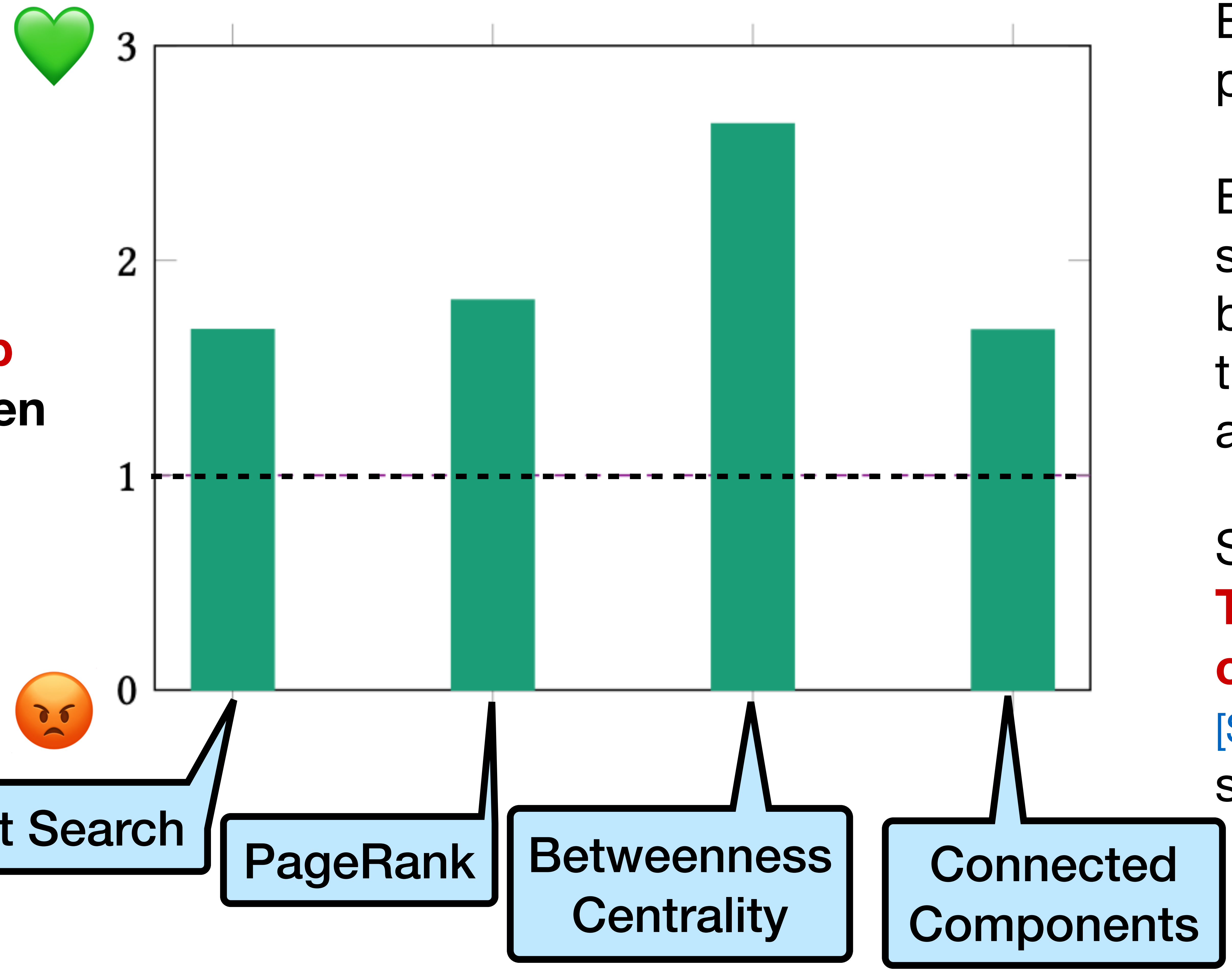
Betweenness centrality
4.72s

Times are **human-measurable** even with parallelism, demonstrating the importance of **efficient processing**

Query Speed in Dynamic Graph Systems

Terrace [PandeyWhXuBu21], a dynamic graph processing system, optimizes further with a **"locality-first design"** that takes advantage of graph structure.

Normalized Speedup of Terrace Over Aspen [DhulipalaShBl19]



Both systems support parallelization.

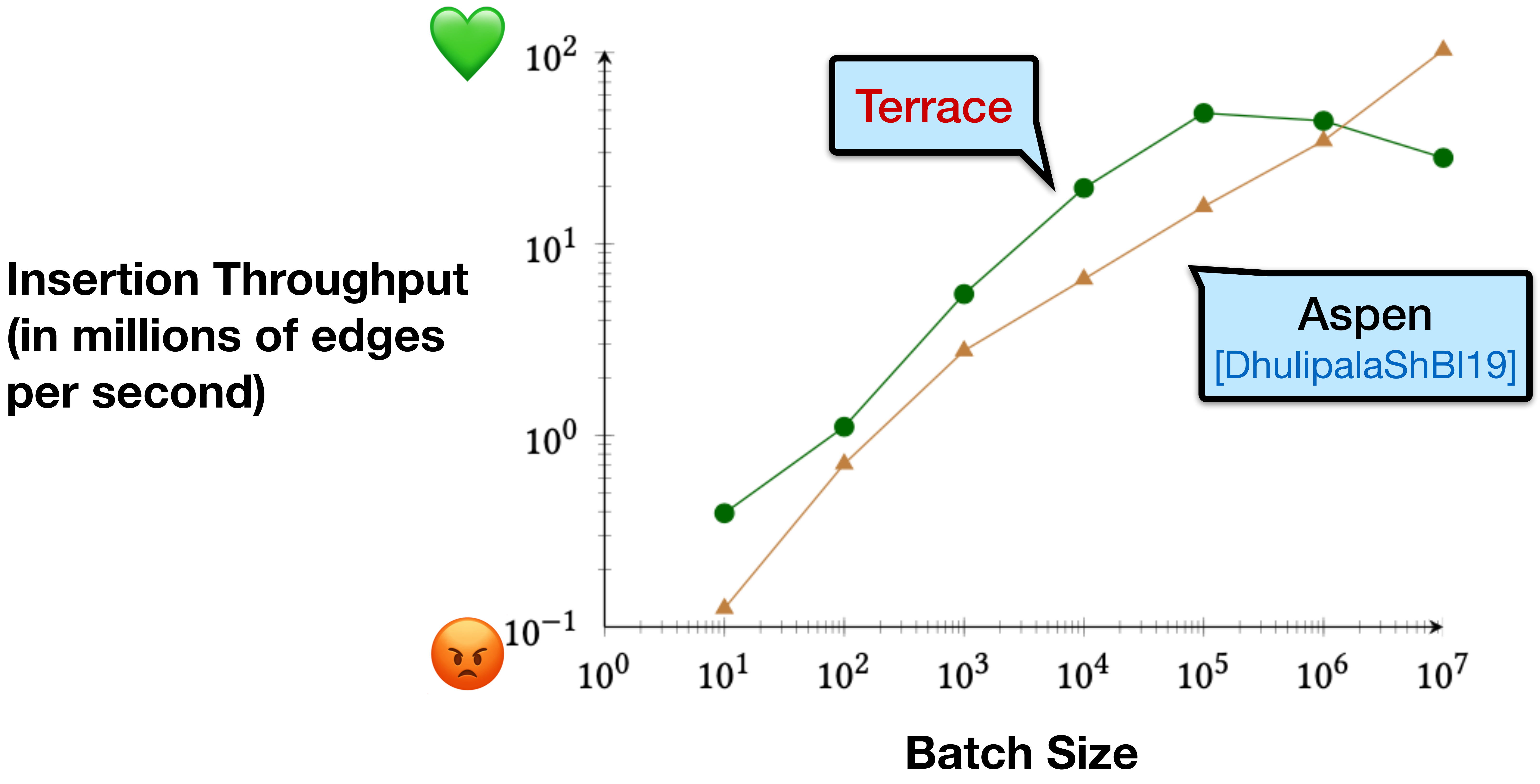
Both systems run the same algorithms by implementing the Ligra [ShunBl13] abstraction.

Surprisingly, in some cases, **Terrace achieves speedup on queries over Ligra** [ShunBl13], a static graph system.

[PandeyWhXuBu21] Pandey, Wheatman, Xu, Buluç. "Terrace: A Hierarchical Graph Container for Skewed Dynamic Graphs." SIGMOD '21.

Updatability of Dynamic Graph Systems

Terrace achieves the **best of both worlds** in query and update performance by taking advantage of locality.



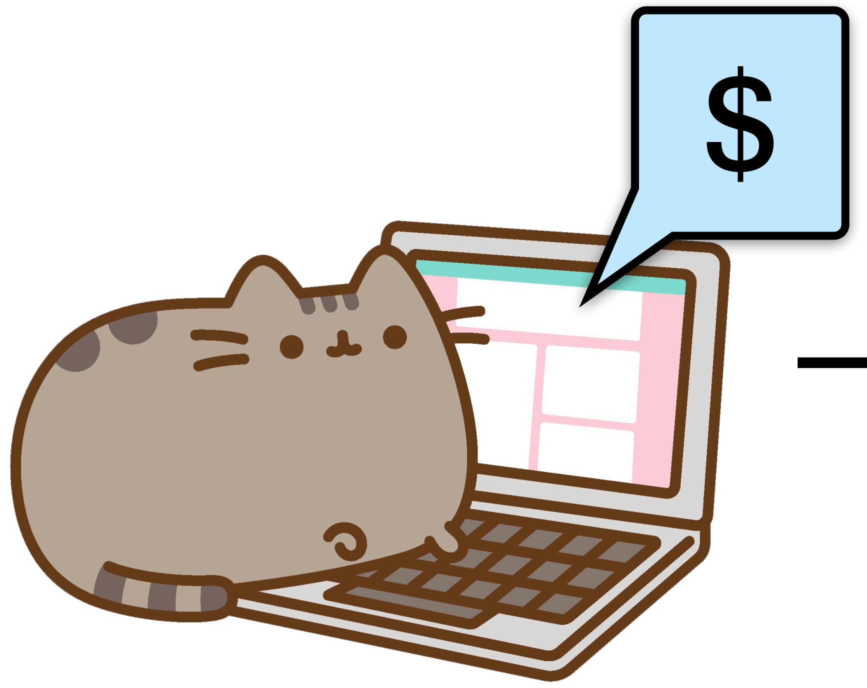
Edges were generated using an rMAT distribution [ChakrabatiZhFa04] and added in batches using the provided API.

Terrace achieves up to 48M inserts per second and up to 9M deletes per second. Future work includes optimizing batch deletions.

Dynamic Graph Processing and the Locality-First Strategy

Problem: Dynamic graph processing

Understand locality in dynamic graph processing



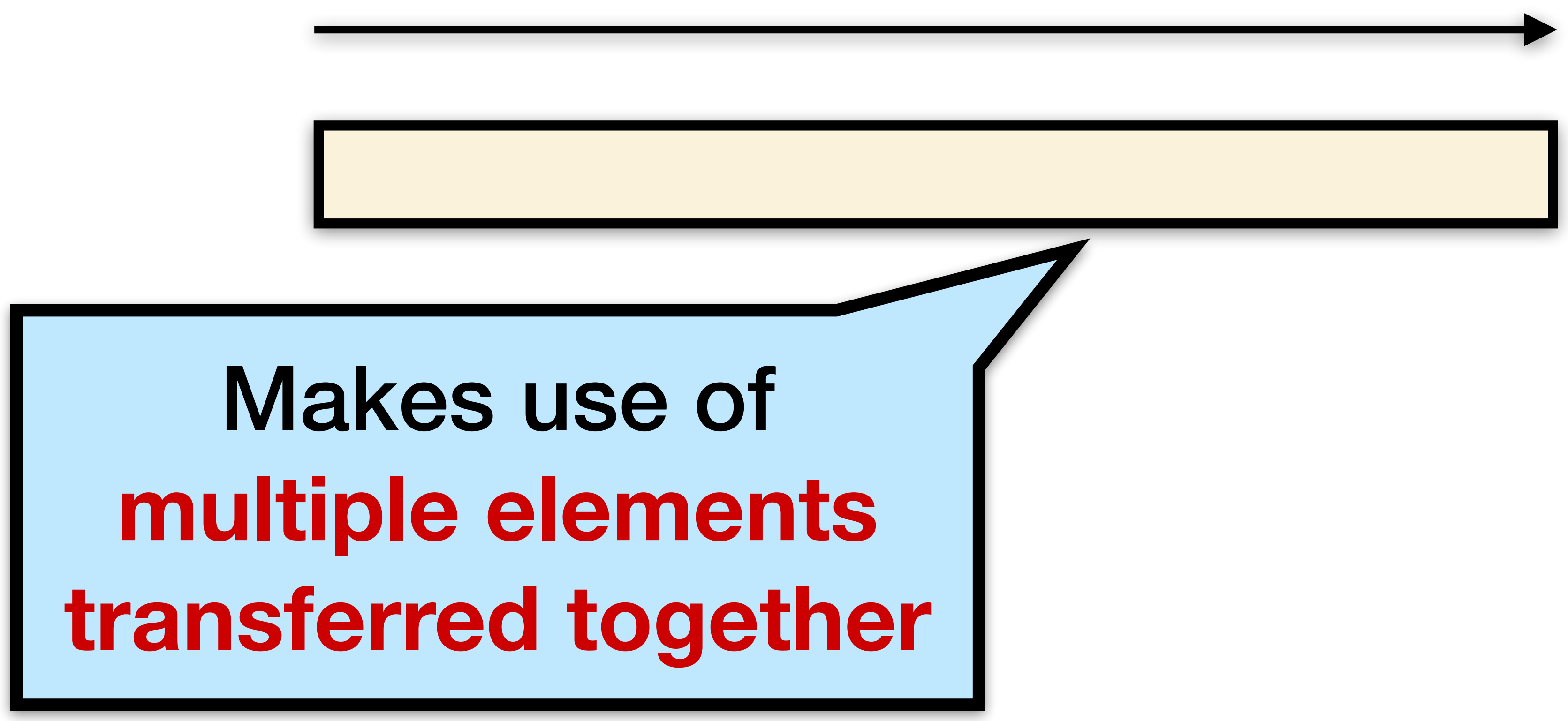
Exploit locality via data structure design for graphs

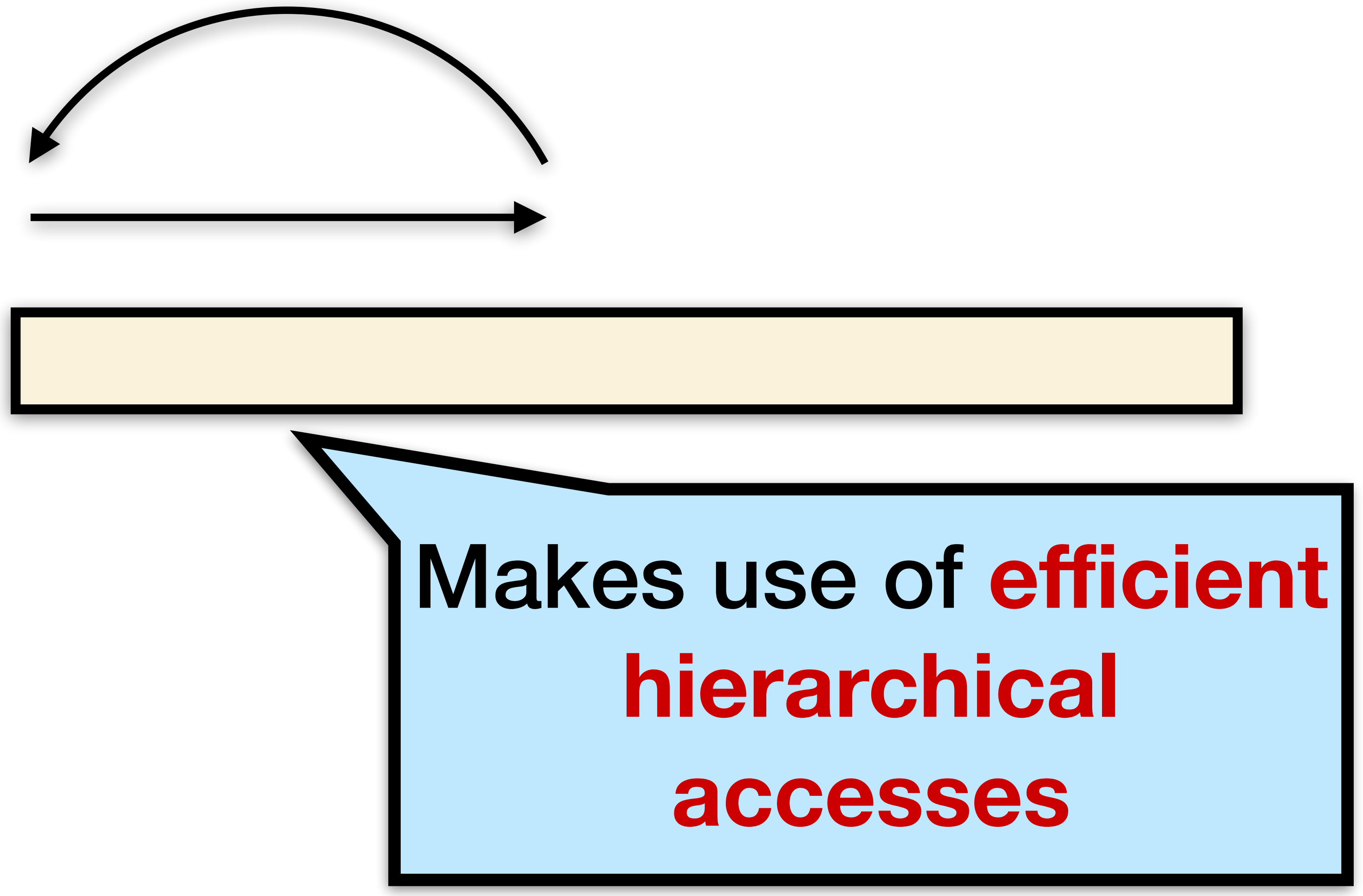
Add **parallelism** into data structures

Two Main Types of Locality

Spatial locality: how many accesses an algorithm makes makes to **nearby** data over a short period of time [Denning72, Denning05].

Temporal locality: how many repeated accesses an algorithm makes to **the same** data over a short period of time [Denning72, Denning05].





Understanding Locality in Graph Queries

Systems for processing dynamic graphs must support fast graph queries.

Vertex scans, or the **processing of a vertex's incident edges**, are a crucial step in graph queries [ShunBl13].

Scan

```
Input: graph G, source vertex src
let Q be a queue
label src as explored
Q.enqueue(src)
while Q is not empty:
  v = Q.dequeue()
  for all edges (v, w) in G.neighbors(v):
    if w not explored:
      label w as explored
```

Breadth-first search

Scan

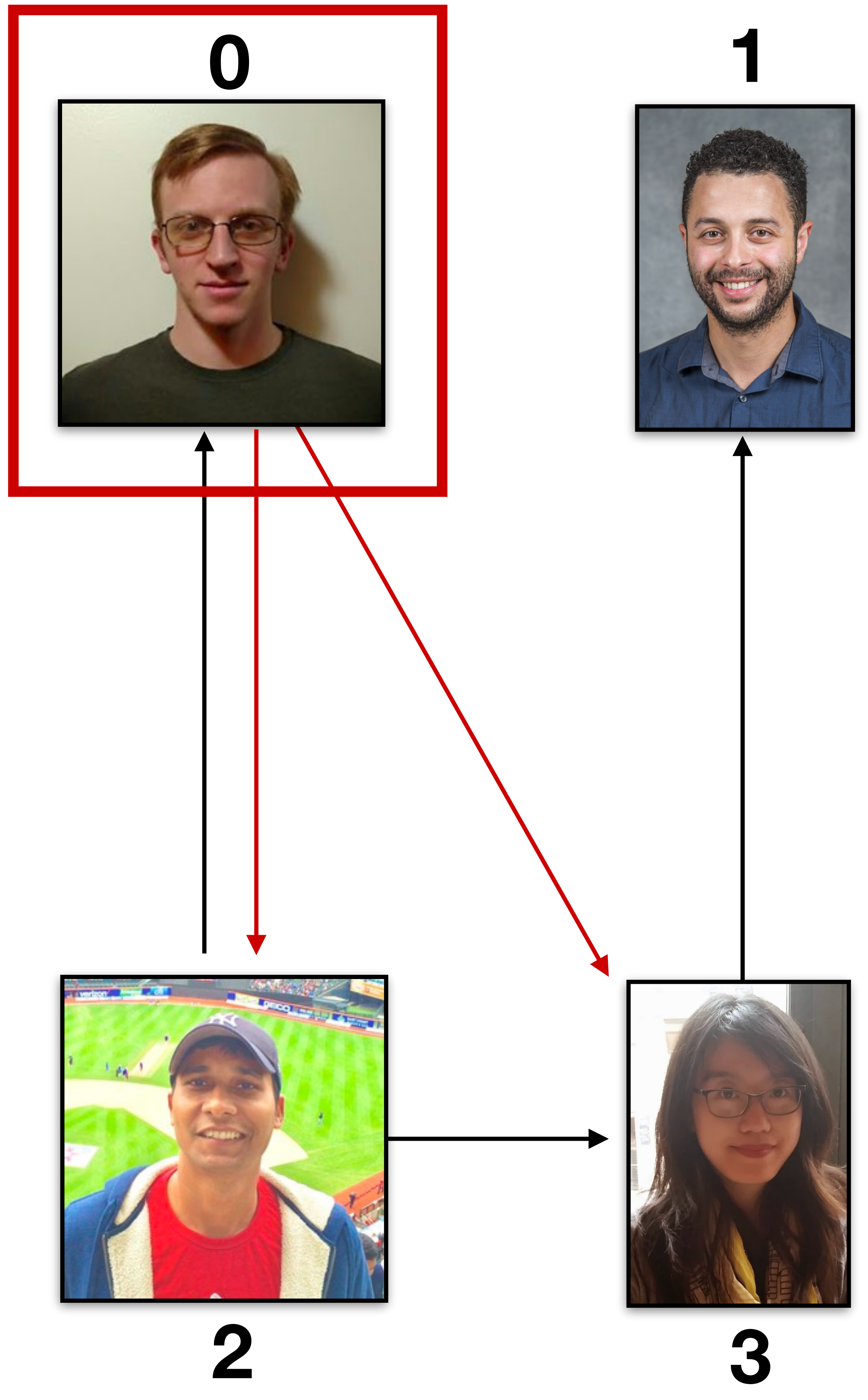
```
Input: graph G
let triangle_count = 0
let E = G.edges()
for (u, v) in E:
  intersect neighbors of u and v:
    if u and v share a neighbor w:
```

Triangle counting

Each neighbor list is scanned at most once (no temporal locality), so **optimize for spatial locality**

Graph Representation Determines Spatial Locality

Scanning a vertex:



Two representations of the neighbors of vertex v ∈ V:

Uncompressed stores a whole row of the adjacency matrix

Sparsity **disrupts locality** with zeroes

0	1	2	3
0	0	1	1

Scan is $\Theta(|V|)$ 😡

Edge array stores neighbors explicitly [EisenstatGuScSh77]

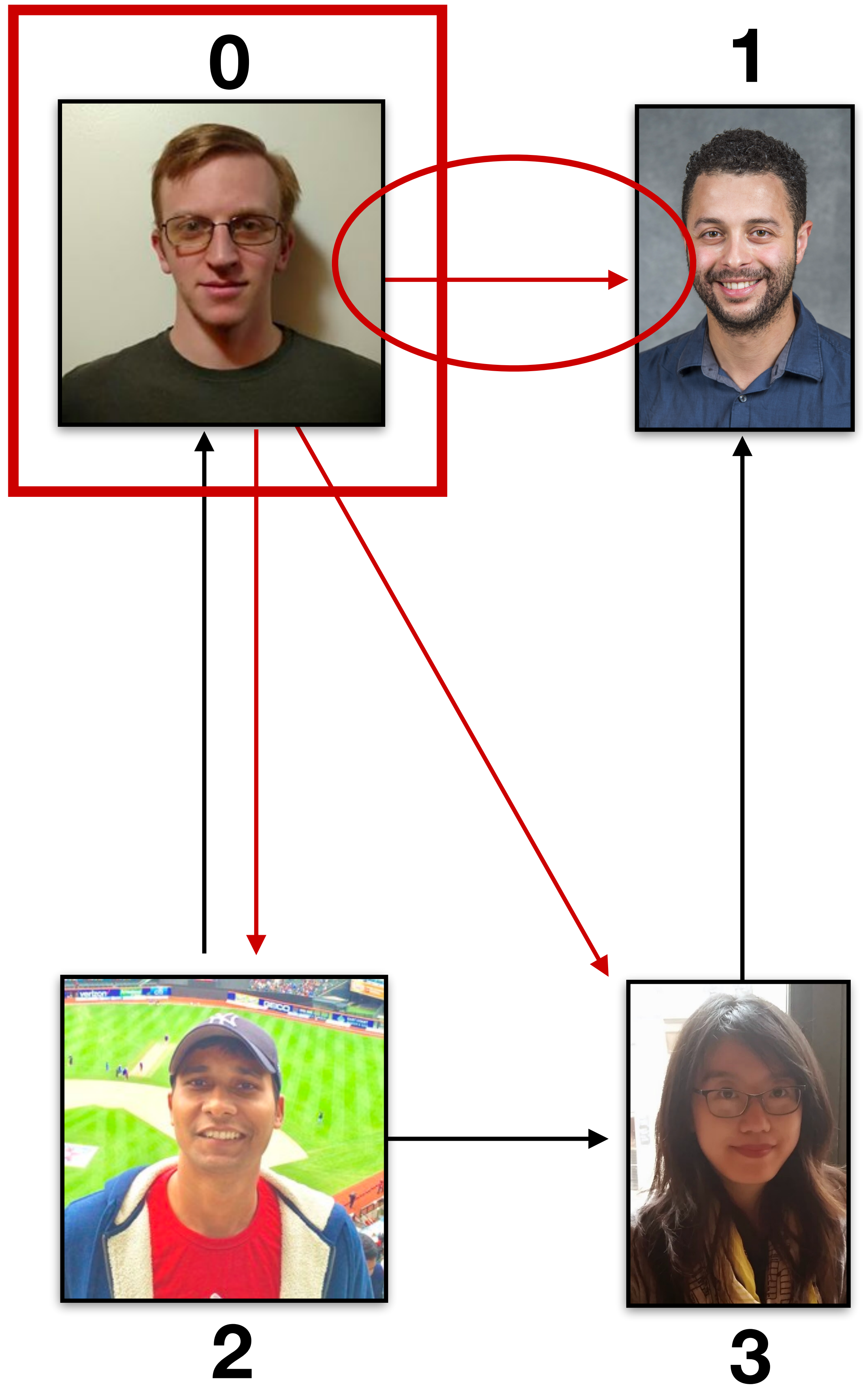
Scan is $\Theta(\text{degree}(v))$ 💚

0	1
2	3

Edge list yields **efficiency savings** on queries by representing and enabling computation on **only the existing edges**

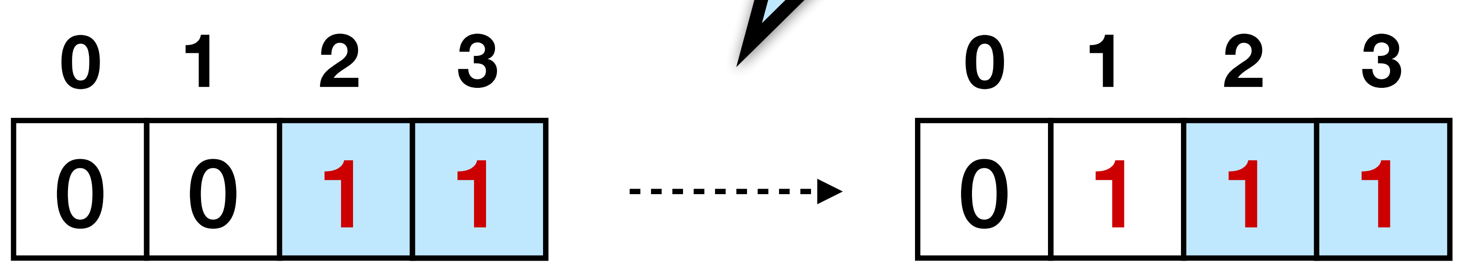
Tensions Between Spatial Locality and Updatability

Updating a vertex:



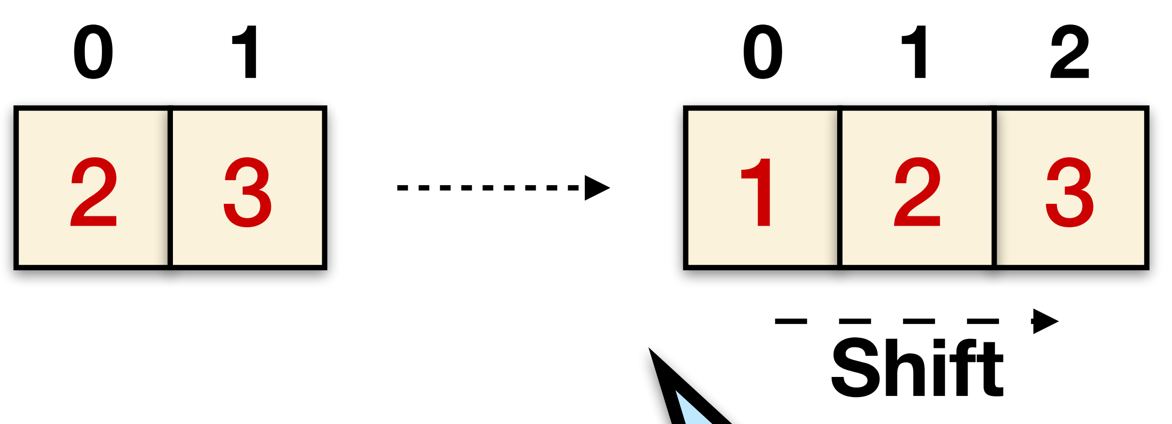
Two representations of the neighbors of vertex v ∈ V:

Uncompressed stores a whole row of the adjacency matrix



Update is O(1)

Edge array stores neighbors explicitly [EisenstatGuScSh77]

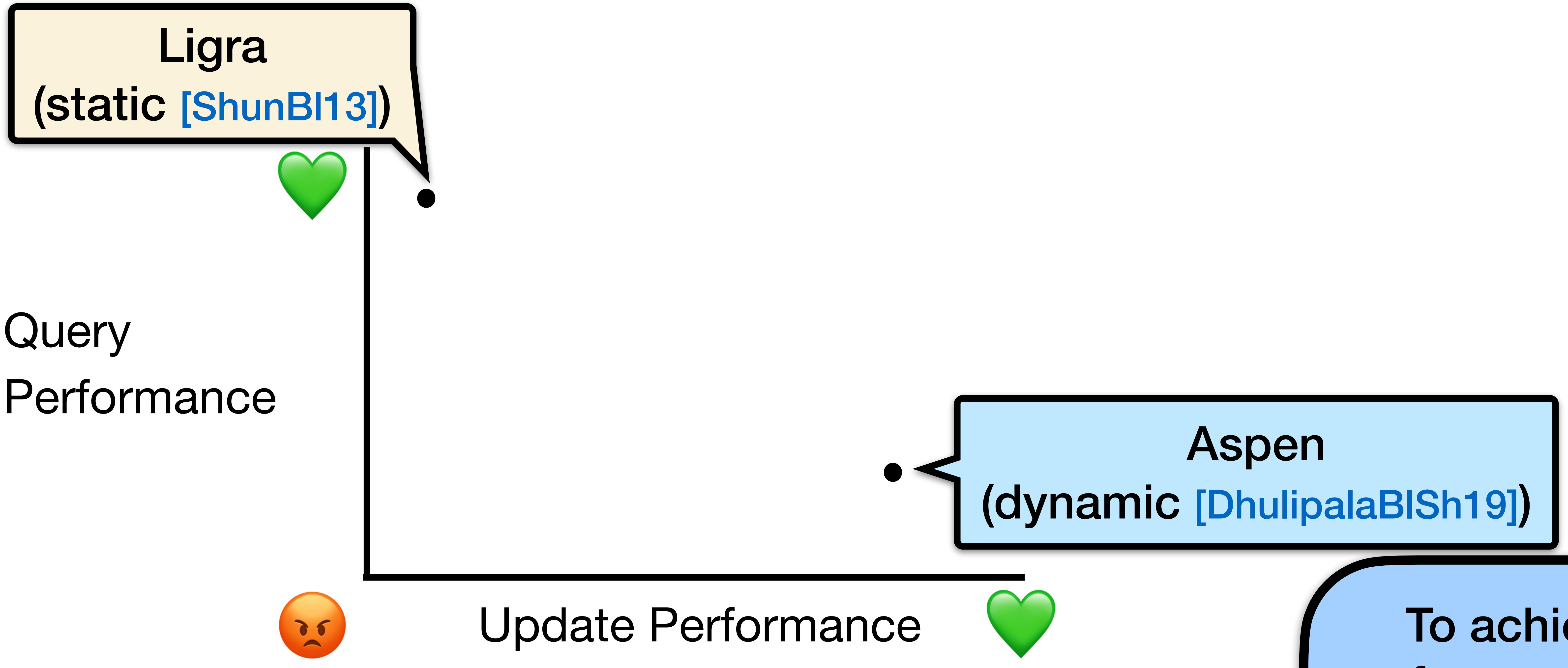


(Ordered) update is $O(\text{degree}(v))$

Problem: can we choose data structures to support efficient scans and updates?

Trading Off Query and Update Performance

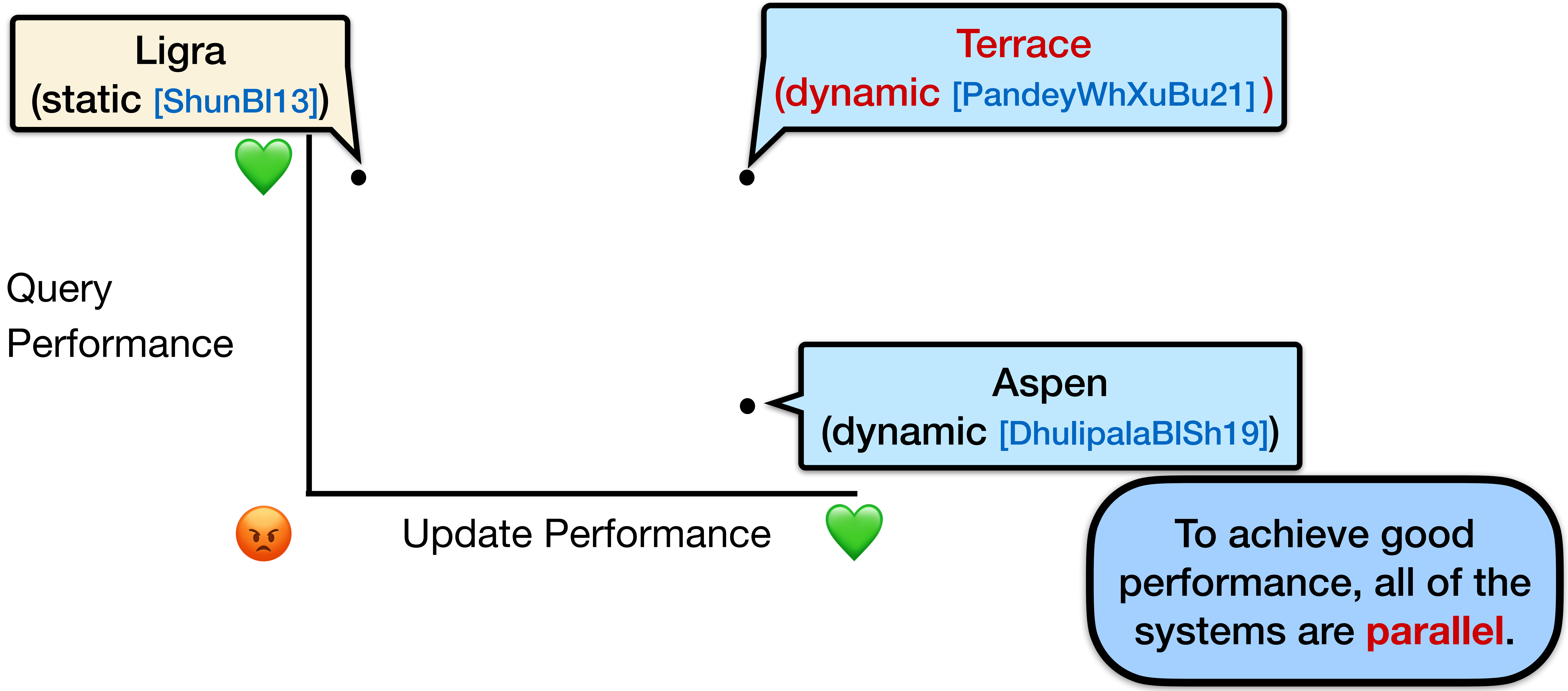
Folk wisdom about graph processing says that **query performance trades off with update performance** [EdigerMcRiBa12, KyrolaBlGu12, ShunBl13, MackoMaMaSe15, DhulipalaBlSh19, BusatoGrBoBa18, GreenBa16] due to **data representation** choices.



To achieve good performance, all of the systems are **parallel**.

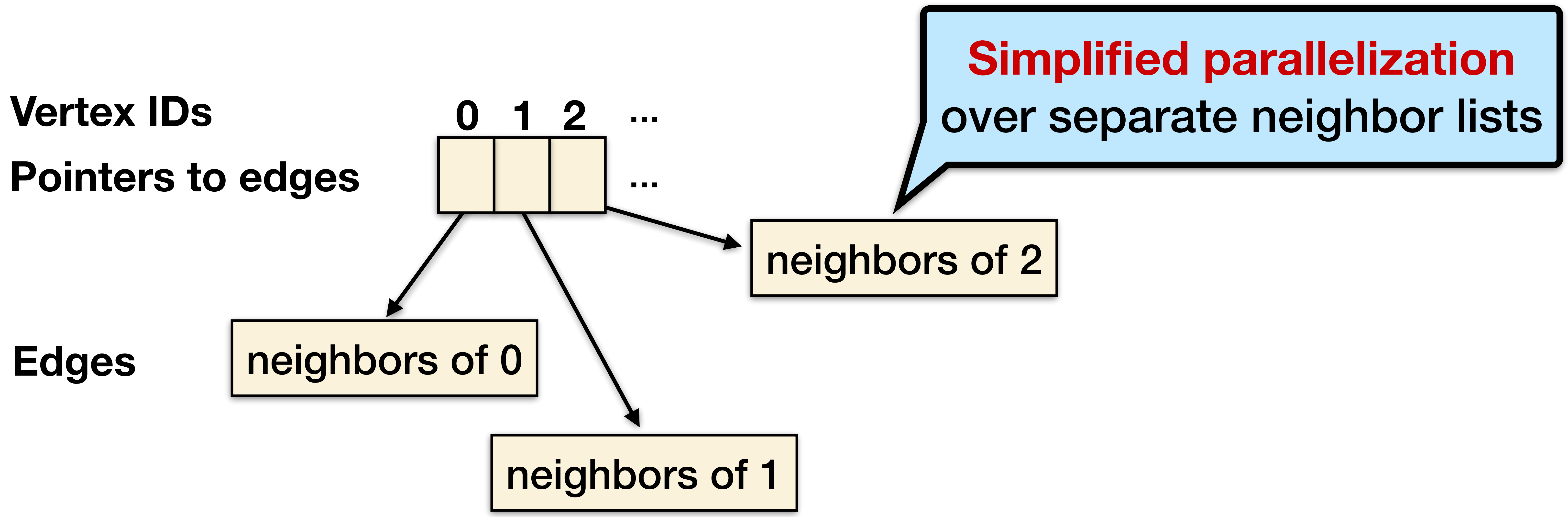
Terrace: A System for Efficiently Processing Dynamic Sparse Graphs

Terrace **overcomes the tradeoff between query and update performance** by using data structures that enhance spatial locality.



Most Graph Systems Separate Neighbor Lists for Parallelization

Existing dynamic graph systems optimize for parallelism first with **separate per-vertex data structures** e.g. trees [DhulipalaBlSh19], adjacency lists [EdigerMcRiBa12], and others [KyrolaBlGu12, BusatoGrBoBa18, GreenBa16].

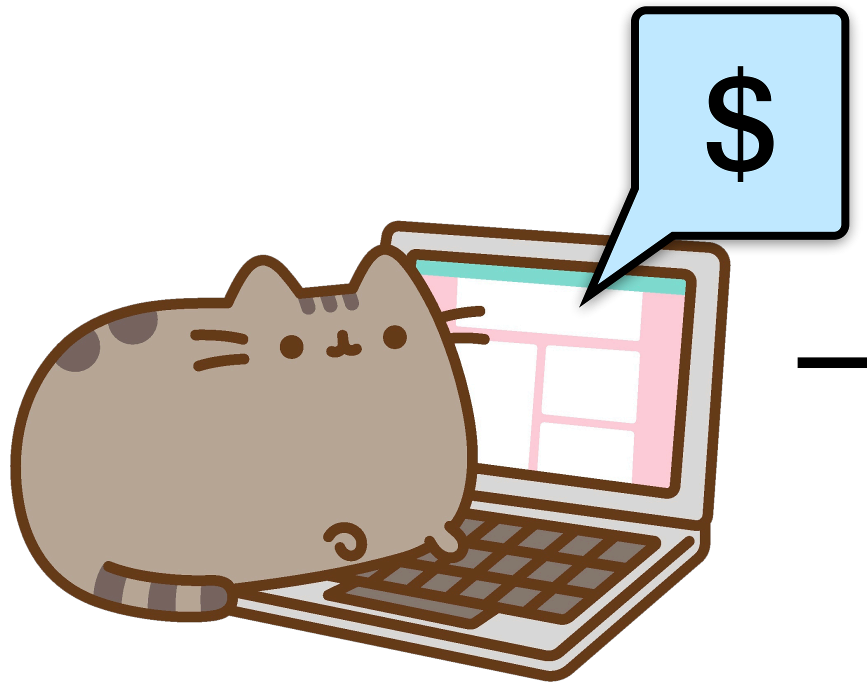


Weakness: Separating the data structures **disrupts locality**.

Dynamic Graph Processing and the Locality-First Strategy

Problem: Dynamic graph processing

Understand locality in dynamic graph processing

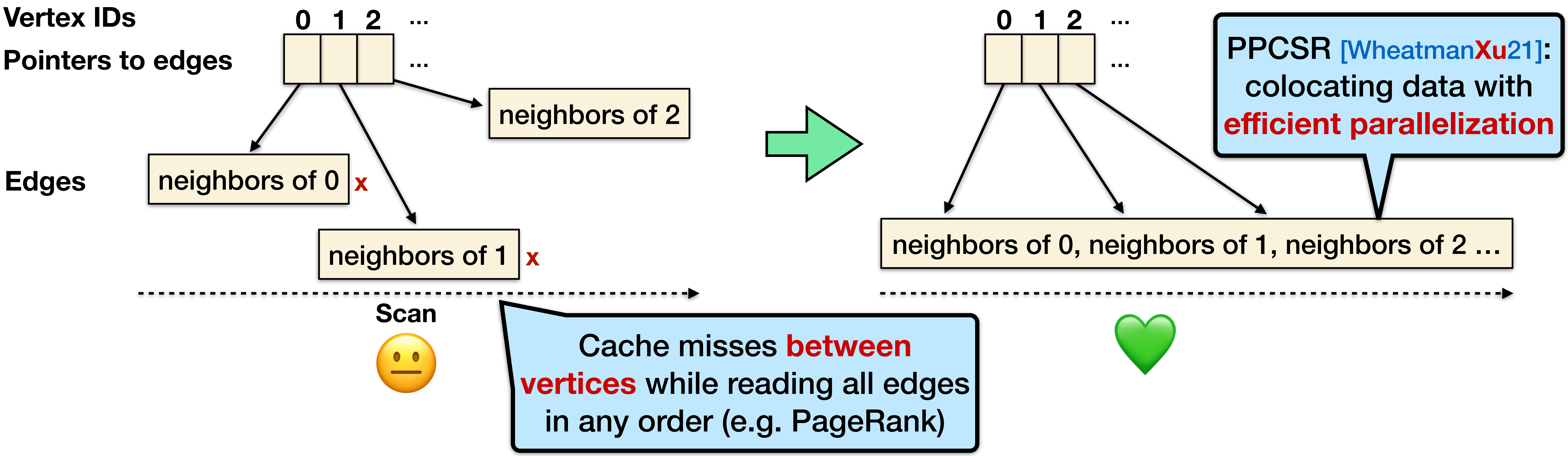


Exploit locality via data structure design for graphs

Add **parallelism** into data structures

Enhancing Spatial Locality by Colocating Neighbor Lists

Idea: Colocate neighbor lists in the same data structure, which **avoids cache misses** when traversing edges in order.



Question: Do these misses actually affect performance, or are they a low-order term?

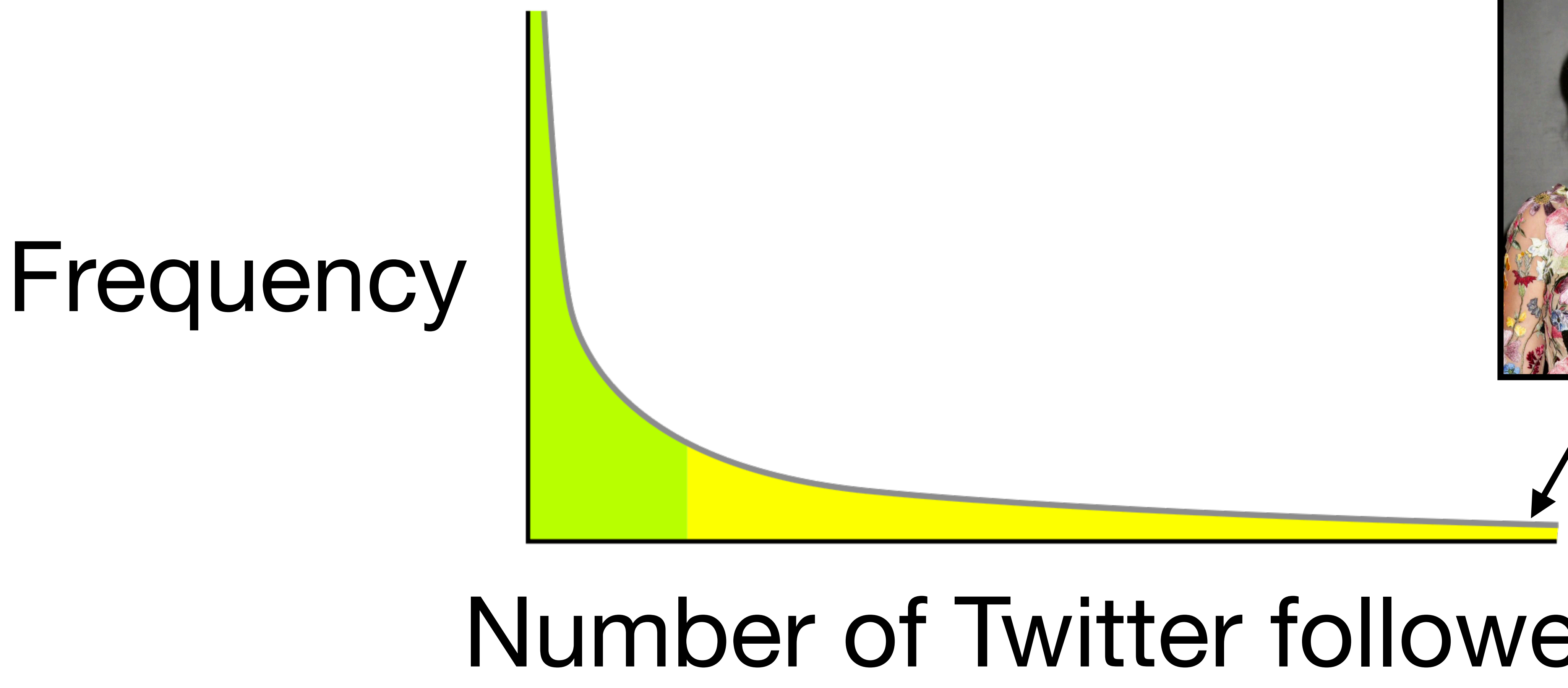
[WheatmanXu21] Wheatman and **Xu**. "A Parallel Packed Memory Array to Store Dynamic Graphs." ALENEX '21.

Dynamic Graphs Are Often Skewed

Real-world dynamic graphs, e.g. social network graphs, often follow a **skewed** (e.g. power-law) distribution with a **few high-degree vertices and many low-degree vertices** [BarabasiAl99].

Example power law:

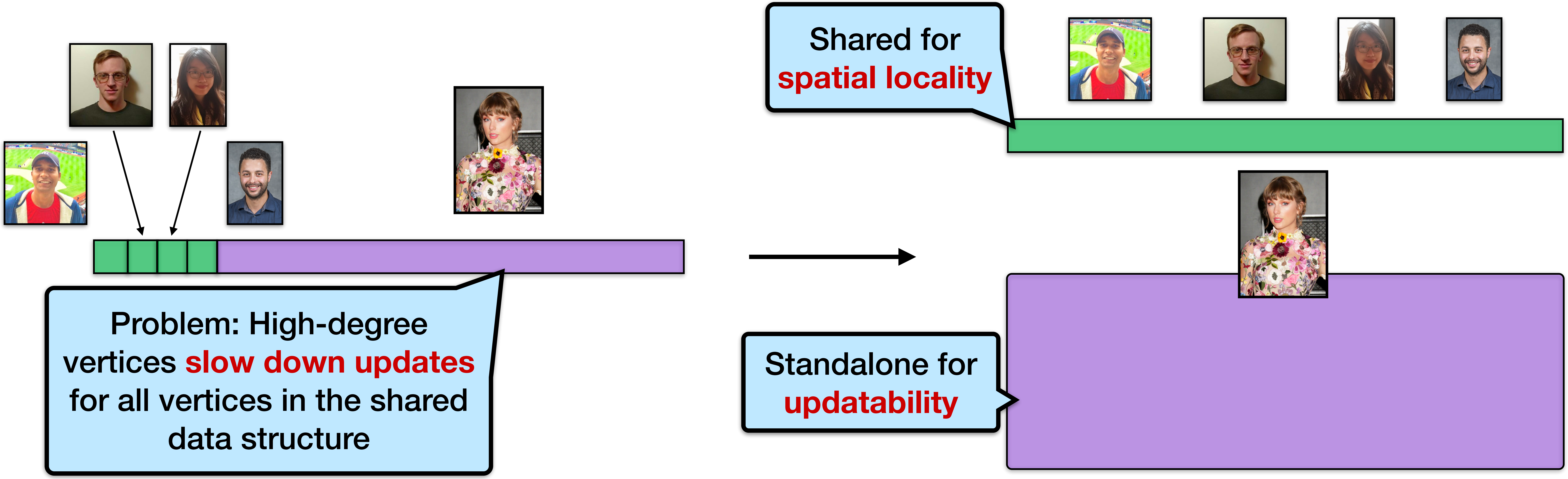
Graph	% < 10 neighbors	% < 1000 neighbors
Twitter	64.56	99.51



These graphs exhibit **high degree variance**: for example, the maximum degree in the Twitter graph is about 3 million [BeamerAsPa15]

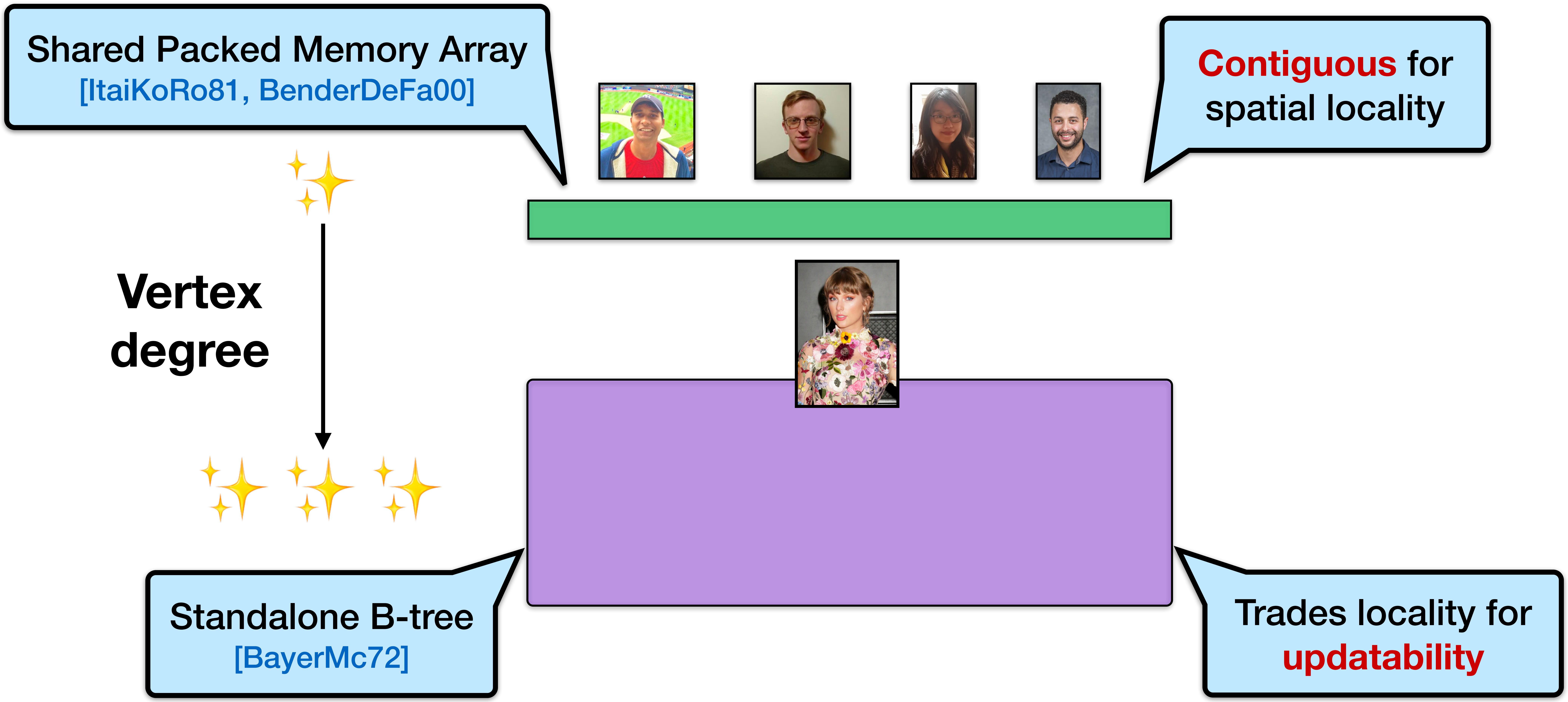
Insight: Locality-First Skew-Aware Design

Next step: **refine the solution** with a **hierarchical design** that takes advantage of skewness while maintaining locality as much as possible.



Implementing the Hierarchical Skew-Aware Design

Terrace implements the locality-first hierarchical design with **cache-friendly data structures.**



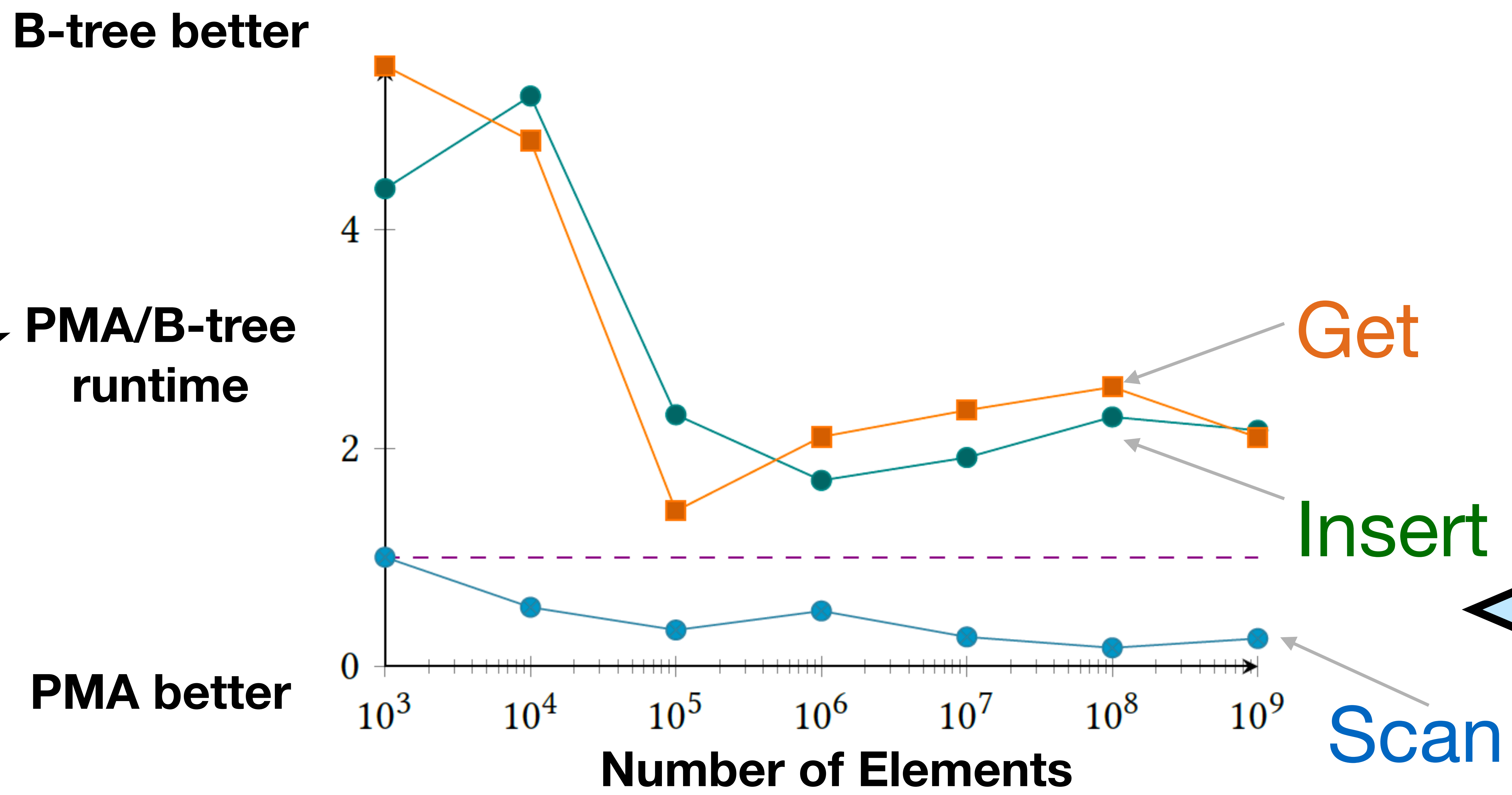
Selecting Data Structures for Dynamic Graphs

In theory, B-trees [BayerMc72] **asymptotically dominate** Packed Memory Arrays (PMA) [ItaiKoRo81, BenderDeFa00] in the classical external-memory model [AggarwalVi88].

Given a cache block size B and input size N, B-trees and PMAs take $\Theta(N/B)$ **block transfers** to scan.

B-tree inserts take $O(\log_B(N))$ transfers, while PMA inserts take $O(\log^2(N))$.

The theory does not capture **sequential vs random access**



Problem: **Neither data structure clearly wins** for dynamic graphs because graphs require fast updates and scans

Solution: **use both**, depending on degree

Exploiting Skewness for Cache-Friendliness

The **locality-first design** in Terrace **reduces cache misses** during graph queries.

Additional optimization: store some edges **in-place for extra spatial locality**

Query	Ligra [ShunBl13]	Aspen [DhulipalaShBl19]	Terrace [PandeyWhXuBu21]
Breadth-first Search	3.5M	6.3M	1.1M
PageRank	174M	197M	128M

Static

Dynamic

On the LiveJournal graph

Cache-friendliness translates into **graph query performance**

Terrace: Applying the Locality-First Strategy to Dynamic Graph Processing

In practice, Terrace is about **2x faster on graph query algorithms** than Aspen while **maintaining similar updatability**.

Terrace's **cache-friendly design** demonstrates the impact of **the locality-first strategy** in graph processing.

Problem: Dynamic graph processing

Understand locality: opportunities for spatial locality due to **skewness**

Exploit spatial locality with a **cache-friendly skew-aware** data structure

Implementation of Terrace, a **parallel** dynamic-graph-processing system based on the **skew-aware design** [PandeyWhXuBu21]

https://github.com/PASSIONLab/terrace

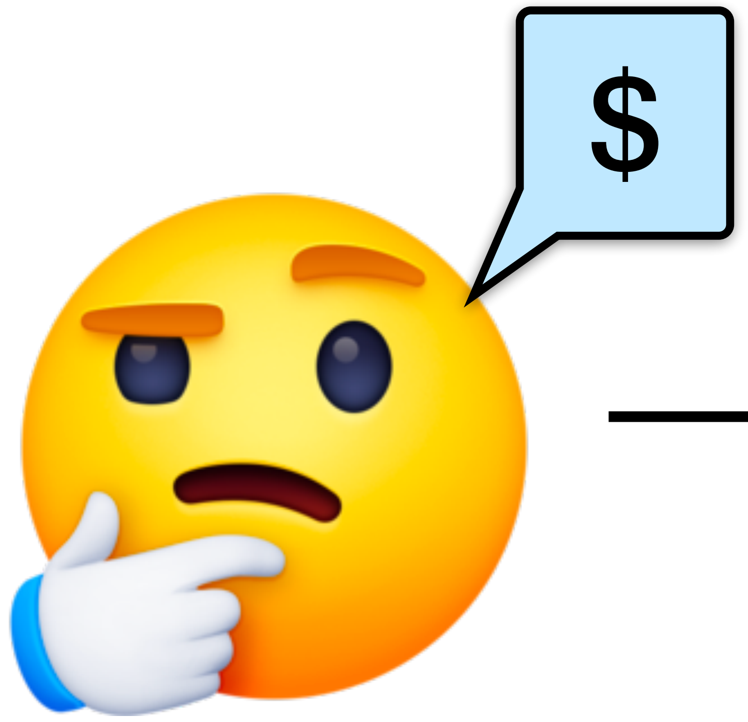
Talk Outline

- ✓ Case Study: Dynamic Graph Processing via the Locality-First Strategy
- ➜ **Applicability of the Locality-First Strategy**
- Other Contributions
- Research Mission, Future Work, and Research Vision

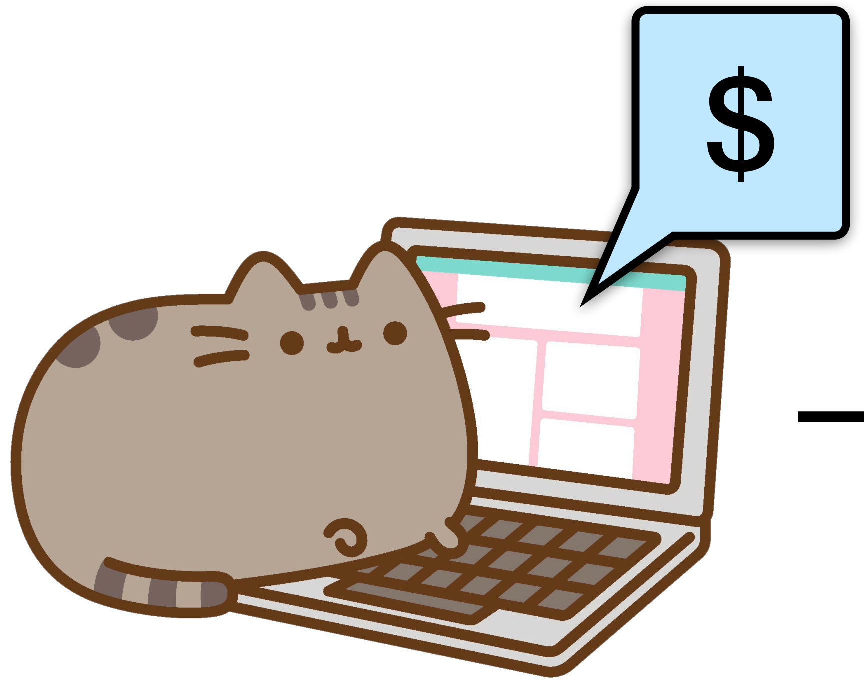
How To Develop Efficient Multicore Algorithms

To create parallel algorithms and data structures for multicores that are theoretically and practically efficient, practitioners should use a **locality-first strategy**.

Problem



Understand locality



Exploit locality via algorithm engineering

Add parallelism

Why locality-first for general problems?

Locality-First Enables Easier Algorithm Engineering

The locality-first strategy **simplifies writing parallel code** by focusing on the serial execution first.

Multithreading

Theory **Practice**

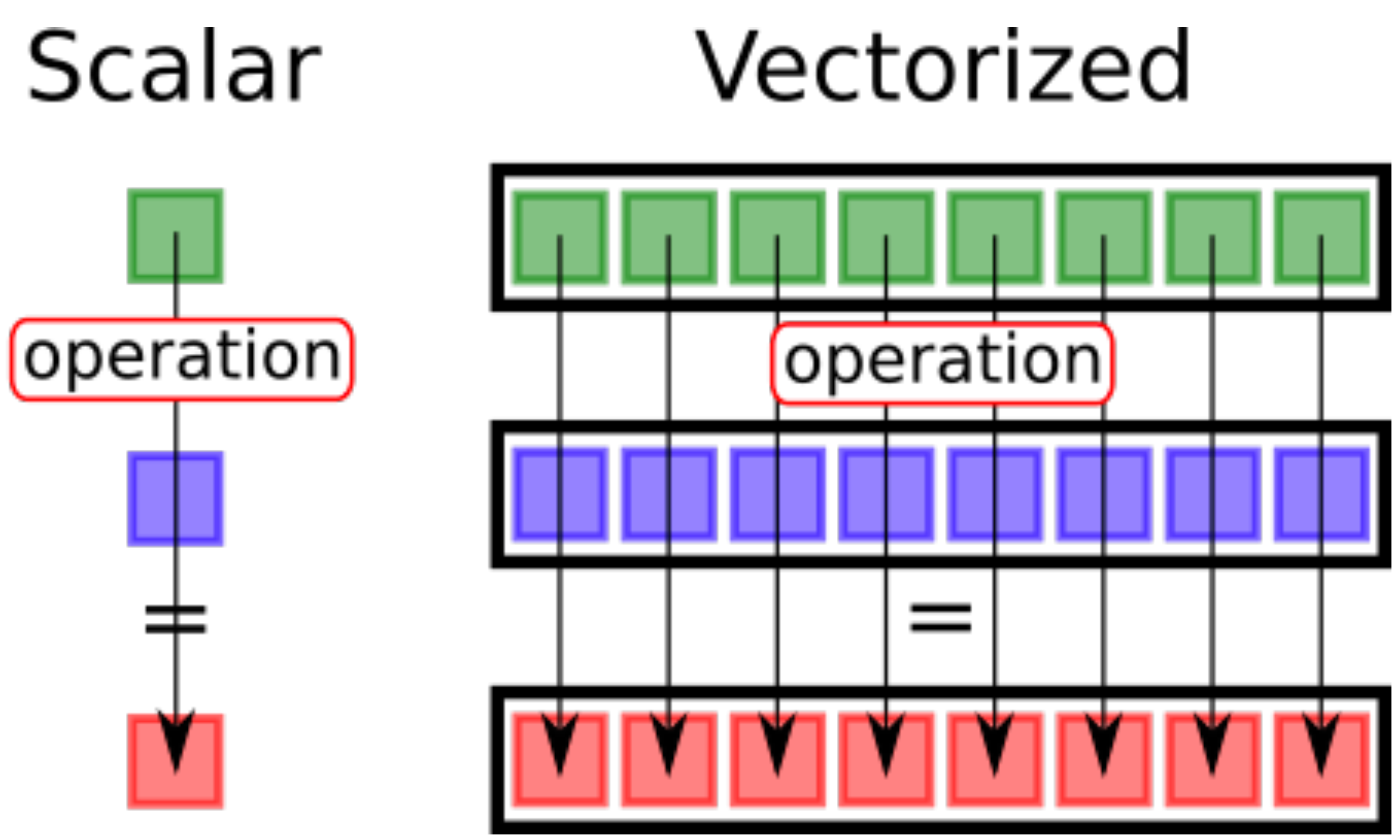
E.g. Race conditions [FengLe97], false sharing [TorrellasLaHe94], profiling scalability [SchardlKuLeLeLe15].

For example, my coauthors and I implemented a (serial) Packed Memory Array [WheatmanXu18] before the parallel version [WheatmanXu21], which Terrace [PandeyWhXuBu21] builds on.

[WheatmanXu18] Wheatman and Xu. “Packed Compressed Sparse Row: A Dynamic Graph Representation.” HPEC ’18.

Spatial Locality Enables Other Types of Parallelism

The locality-first strategy draws inspiration from Cilk's [FrigoLeRa98] work-first principle of **minimizing the work in serial**, allowing for peak efficiency after task parallelization.



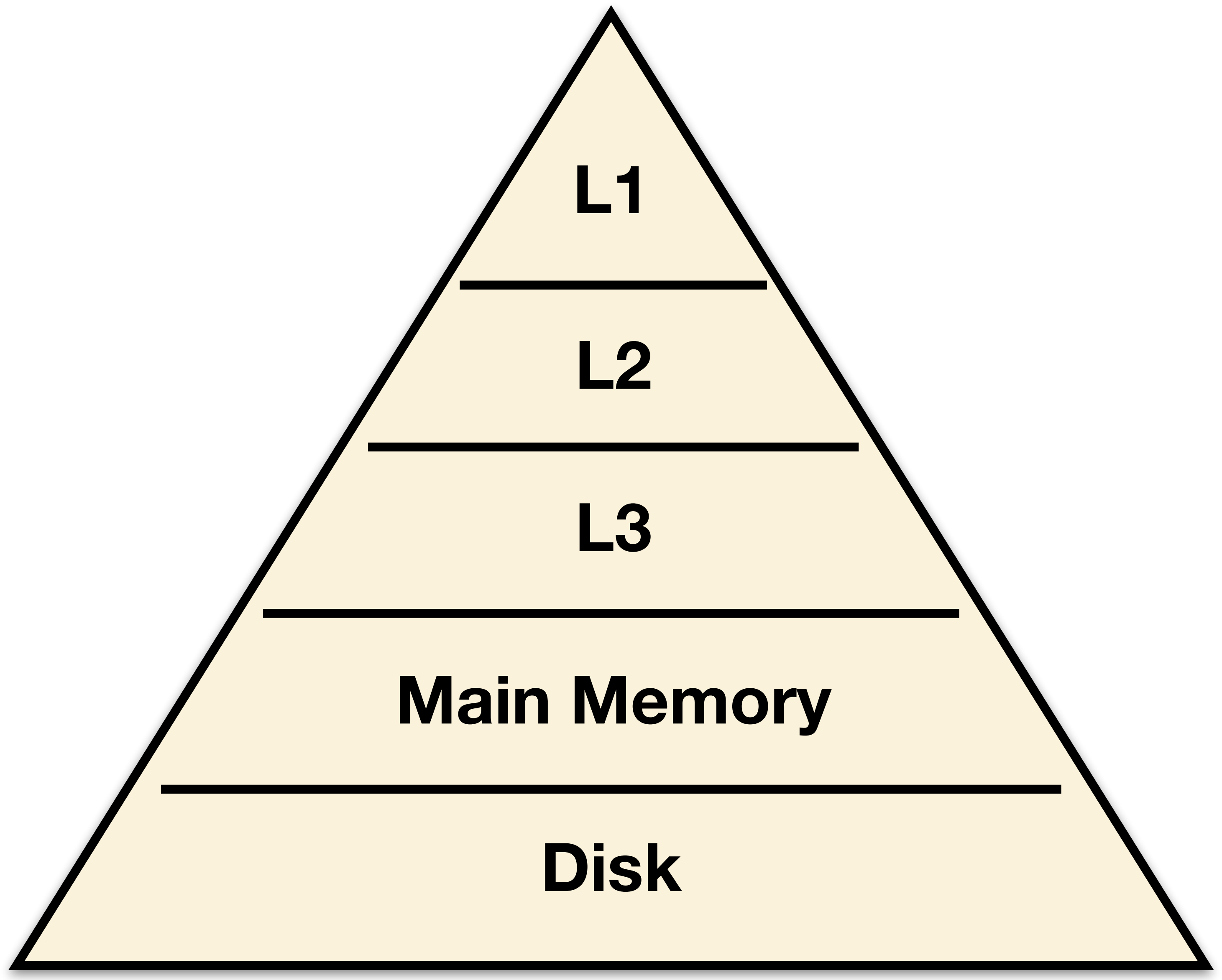
Applying the same operation to a set of values

Data-Level Parallelism
(e.g. SIMD [Flynn72])

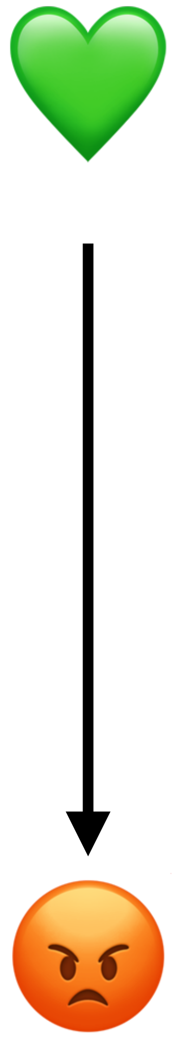
e.g. vectorizing the PMA in Terrace is more straightforward because it is contiguous

Temporal Locality Offers Multiple Opportunities for Performance Improvement

In reality, speedups due to temporal locality are **continuous** because of the multiple levels of the cache hierarchy.



Numbers Everyone Should Know
[Dean09]



0.5ns

7ns

25ns (approx.)
[Intel i7 guide]

100 ns

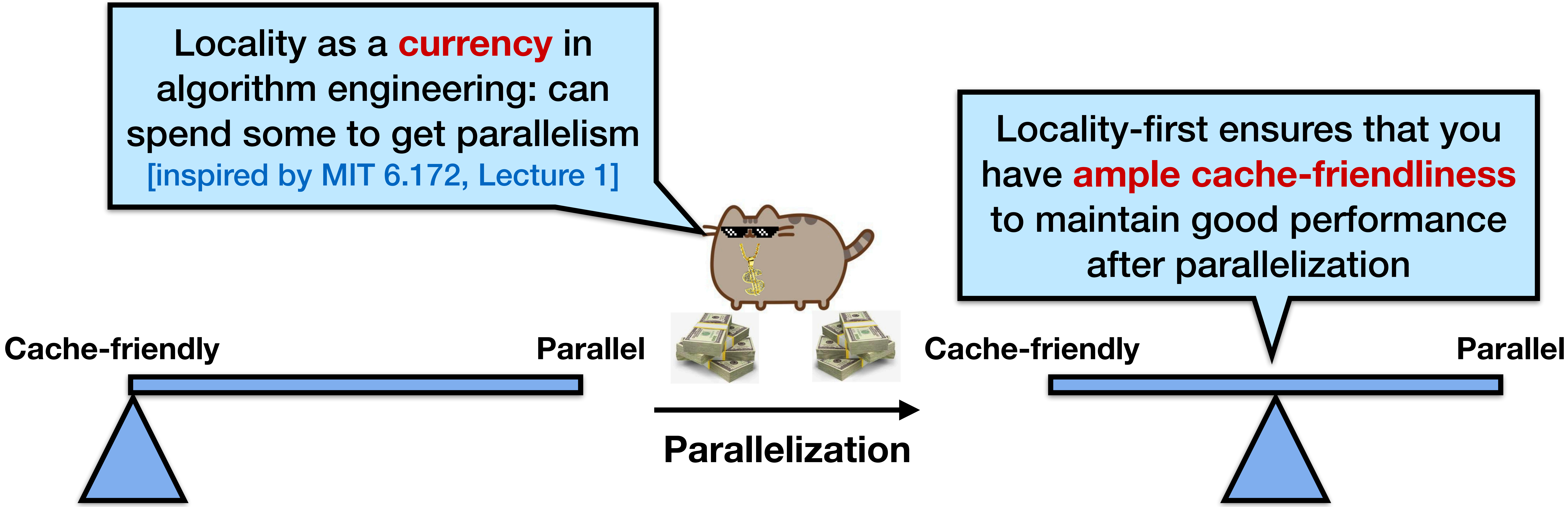
10,000,000 ns

My work [Bender ***et al.*** 20, LincolnLiLy**Xu**18] touches on cache-oblivious algorithms [FrigoLePrRa99], which use all levels of cache asymptotically optimally.

[Bender *et al.* 20] Bender ***et al.*** "Closing the Gap Between Cache-oblivious and Cache-adaptive Analysis." SPAA '20.
[LincolnLiLyXu18] Lincoln, Liu, Lynch, **Xu**. "Cache-Adaptive Exploration: Experimental Results and Scan-Hiding for Adaptivity." SPAA 18.

Balancing Parallelism and Cache-Friendliness

The locality-first strategy may be surprising for overall performance improvement because **locality and parallelism conflict** with each other.

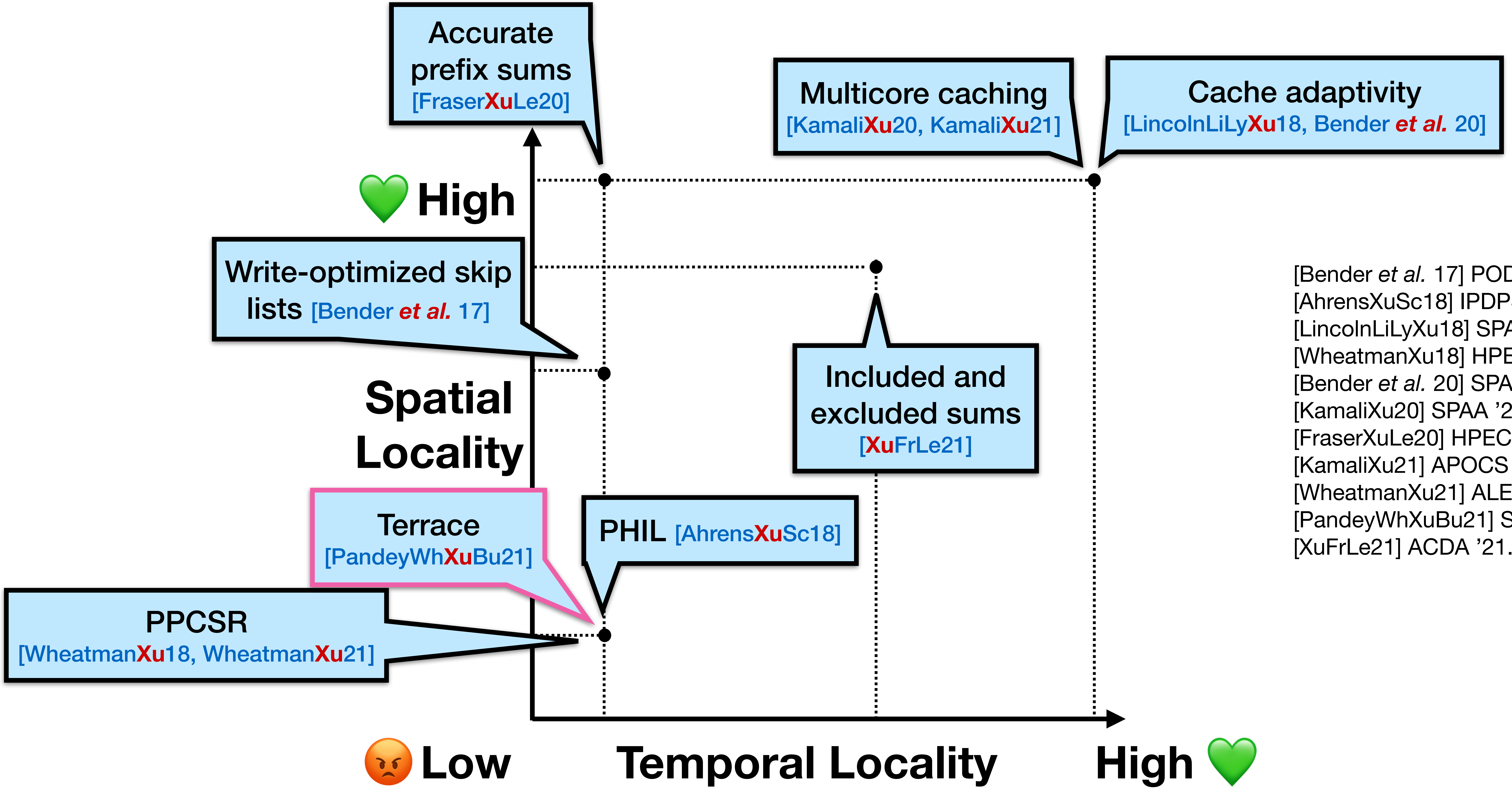


For example, Terrace optimizes for locality first and then trades some of it for efficient parallelization.

Talk Outline

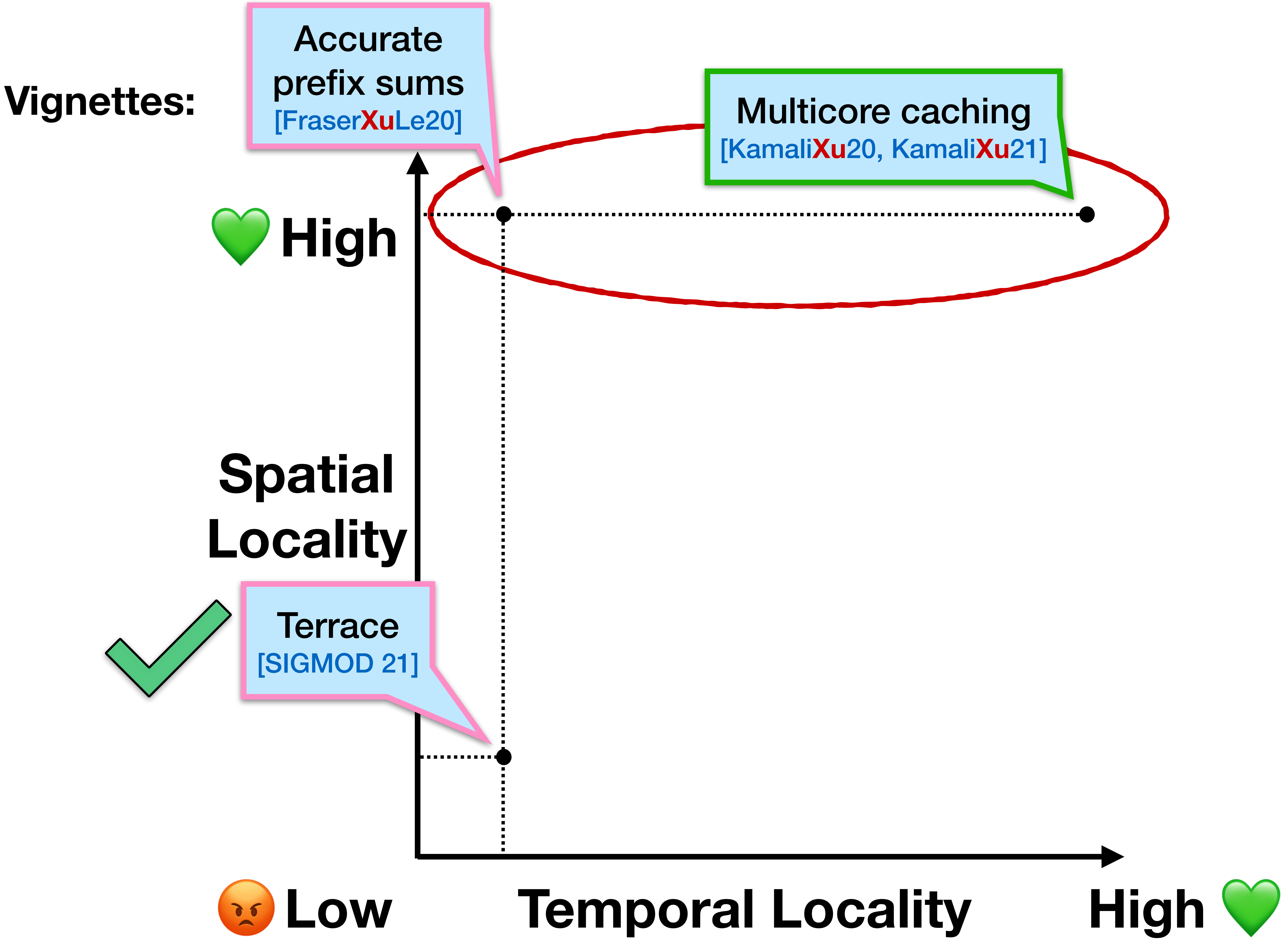
- Case Study: Dynamic Graph Processing via the Locality-First Strategy
- Applicability of the Locality-First Strategy
- **Other Contributions**
- Research Mission, Future Work, and Research Vision

Classification of Contributions



[Bender *et al.* 17] PODS ’17.
[AhrensXuSc18] IPDPS ’18.
[LincolnLiLyXu18] SPAA ’18.
[WheatmanXu18] HPEC ’18.
[Bender *et al.* 20] SPAA ’20.
[KamaliXu20] SPAA ’20.
[FraserXuLe20] HPEC ’20.
[KamaliXu21] APOCS ’21.
[WheatmanXu21] ALENEX ’21.
[PandeyWhXuBu21] SIGMOD ’21.
[XuFrLe21] ACDA ’21.

Exploring the Locality-First Strategy



Example Problem: Accurate Prefix Sums

Prefix sums (aka scans) appear in a wide range of applications and have been targeted for **efficient implementations** e.g. Parlaylib [BlellochAnDh20], NVIDIA GPU [HarrisSeOw07].

Input array of elements

Output array of running sums

$$y_k = \begin{cases} x_0 & \text{if } k = 0 \\ x_k + y_{k-1} & \text{if } k \geq 1. \end{cases}$$

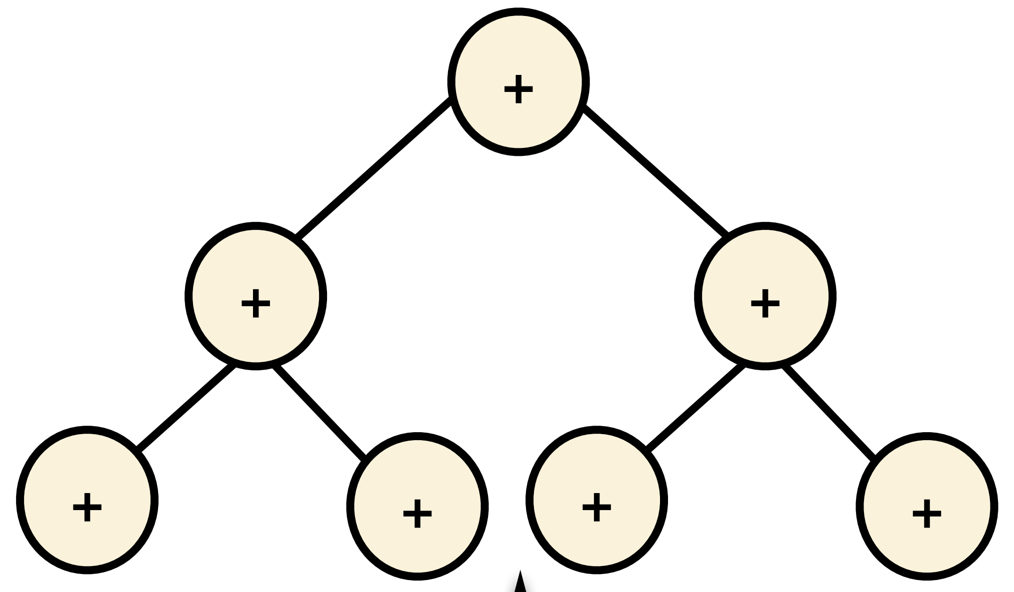
Floating-point prefix sums underlie applications in scientific computing such as summed-area table generation [Crow84] and the fast multipole method [GreengardRo85].

Example: Locality-First in Accurate Prefix Sums

Limited machine precision

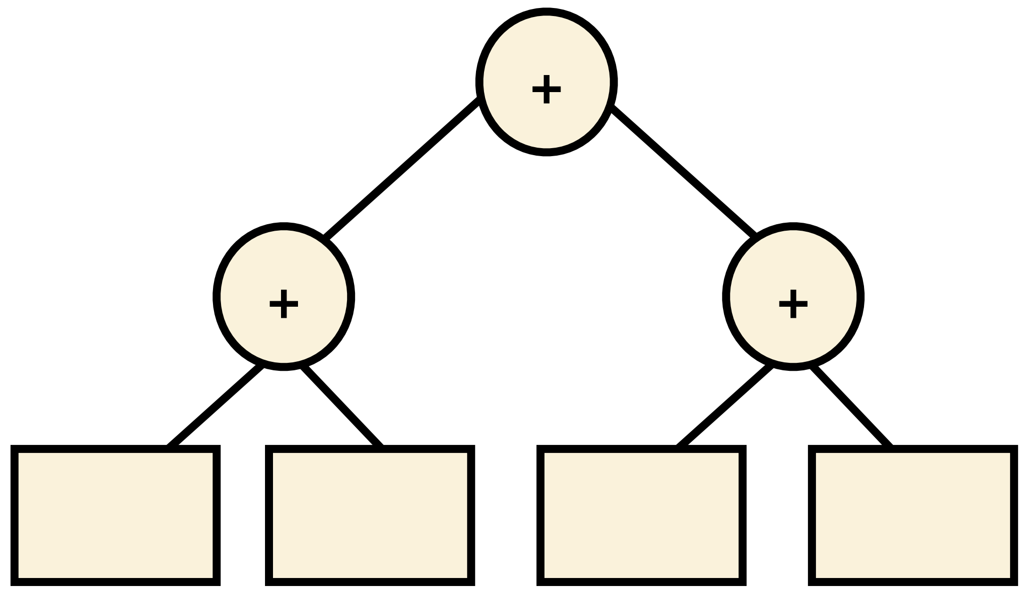
Problem: Minimize error in fast floating-point prefix sums [Higham93].

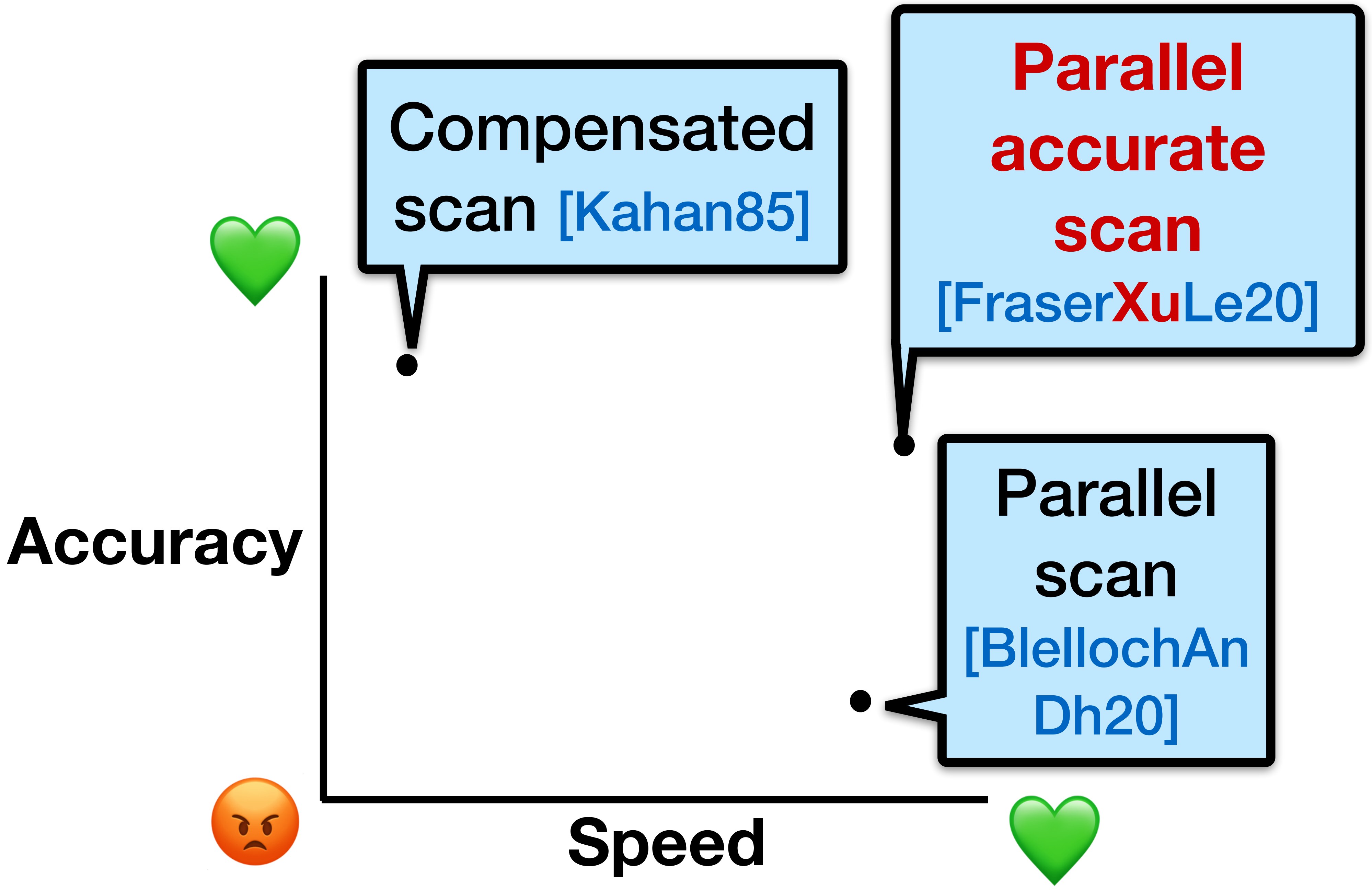
Understand locality:
tree summation has limited temporal locality, opportunities for spatial

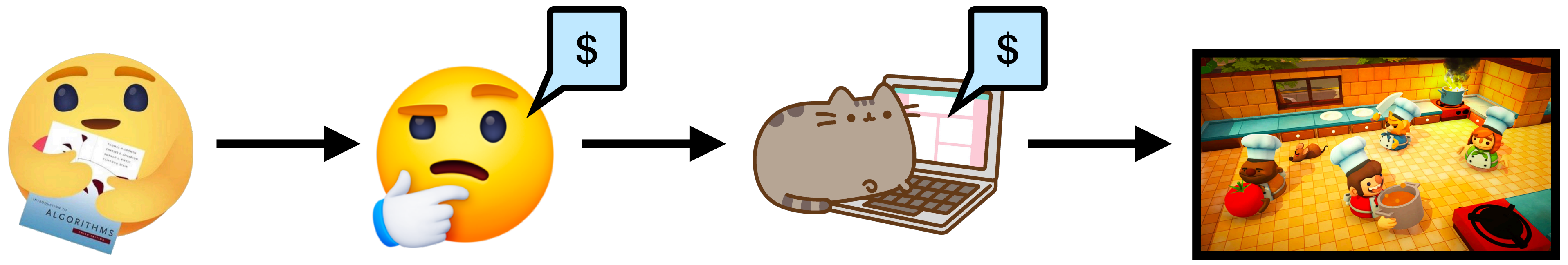


Tree-like summation
to reduce error

Exploit locality:
tree blocking for spatial locality







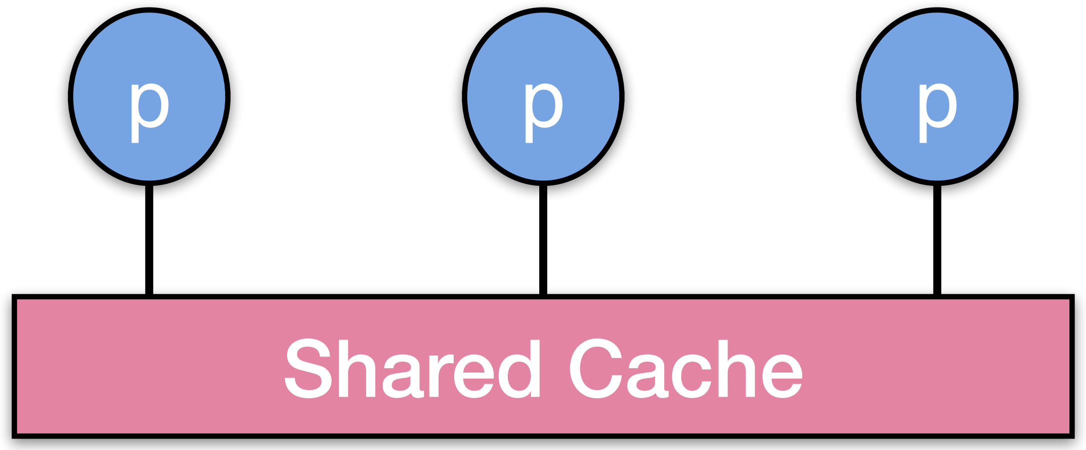
[FraserXuLe20] Fraser, **Xu**, Leiserson. “Work-Efficient Parallel Algorithms for Accurate Floating-Point Prefix Sums.” HPEC ’20.

Example Problem: Multicore Cache Replacement

One possible concern with locality-first is that locality and parallelism are in **tension** with one another.

For example, every multicore system with shared memory must implement a **cache replacement policy** that decides what to evict when the cache gets full.

Parallelism can **disrupt cache-friendliness** of cache-replacement algorithms when multiple workers contend for space [López-OrtizSa12, KattiRa12].



Example: Grounding Locality-First in Multicore Cache Replacement

Goal: **Theoretically ground** the locality-first strategy in multicore cache replacement via a new theoretical framework that extends "beyond-worst-case" analysis to take **temporal locality** into account [Roughgarden20]

Multicore Caching [López-OrtizSa12]	**Worst-case analysis** [KamaliXu20]	**Cyclic analysis** [KamaliXu21]
Least-Recently-Used [SleatorTa84]	😡	💚
Anything else	😡	😡

Grounds the **empirical superiority of LRU** due to naturally-occurring locality [AlbersFaGi02]

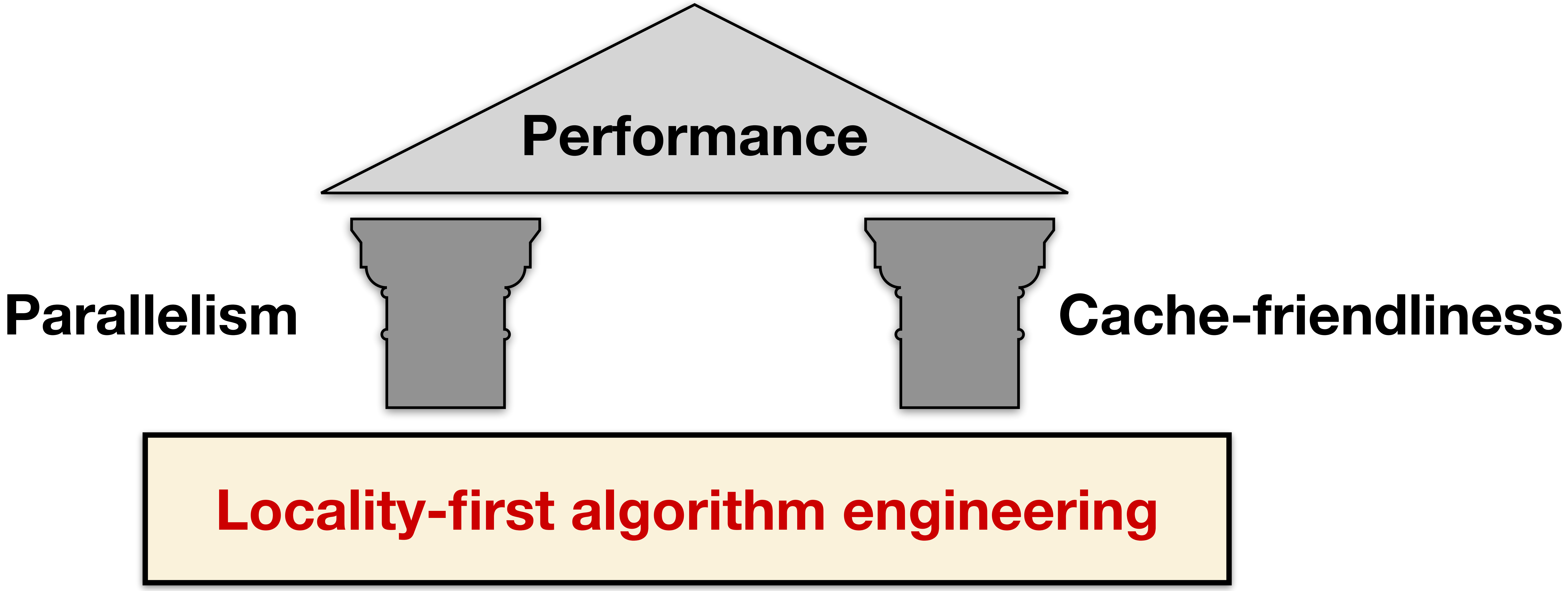
[KamaliXu20] Kamali and **Xu**. "Brief Announcement: Multicore Paging Algorithms Cannot Be Competitive." SPAA '20.
[KamaliXu21] Kamali and **Xu**. "Beyond Worst-case Analysis of Multicore Caching Strategies." APOCS '21.

Talk Outline

- ✓ Case Study: Dynamic Graph Processing via the Locality-First Strategy
- ✓ Applicability of the Locality-First Strategy
- ✓ Other Contributions
- ➡ **Research Mission, Future Work, and Research Vision**

Research Mission

My research mission is to study **algorithms and software technology** to incorporate cache-friendliness and parallelism into applications so that they can easily be optimized.

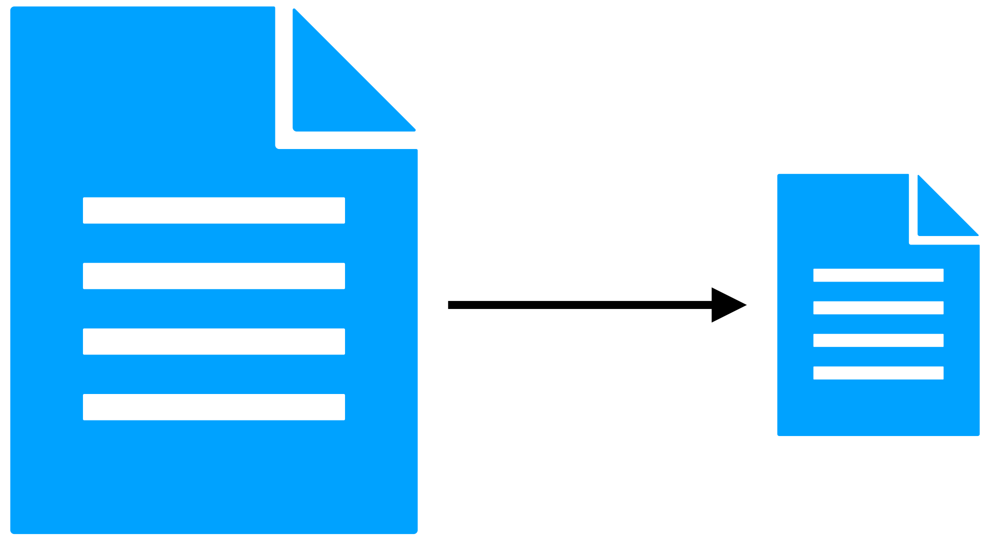


Locality-First in Problems with Low Spatial Locality Via Compression

One direction for future work involves improving spatial locality in sparse problems with **compression**.

Sparse problems

+



Compression
[Smith97]

=

Problem: compression often **trades off with parallelism**

Grounding Locality-First in Problems with Temporal Locality

Another direction involves **beyond-worst-case analysis** of algorithms by taking temporal locality into account.

Multicore cache-replacement algorithm

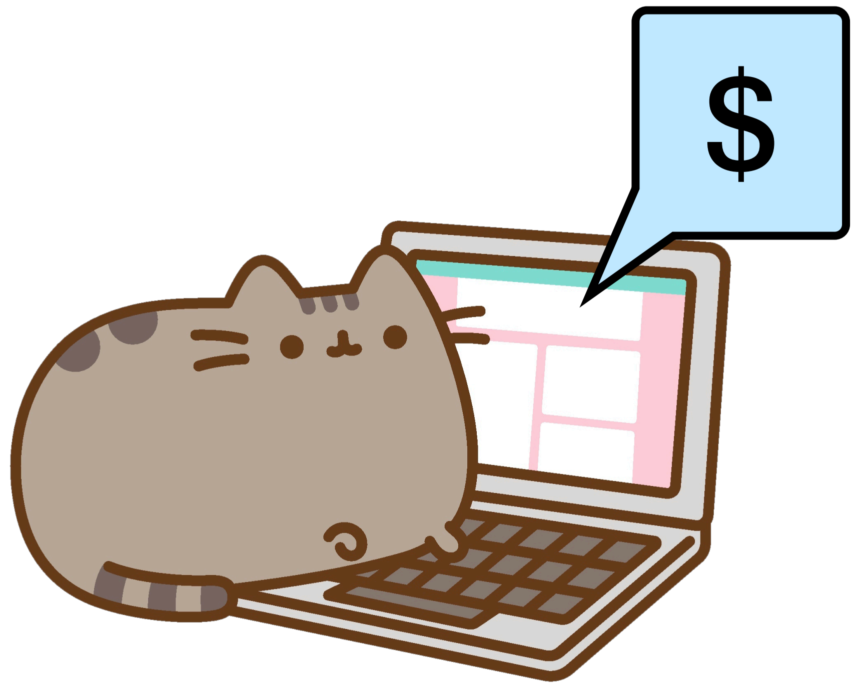
+

Prediction about locality

[LykourisVa18, Rohatgi20]

Some knowledge of **future accesses**

=

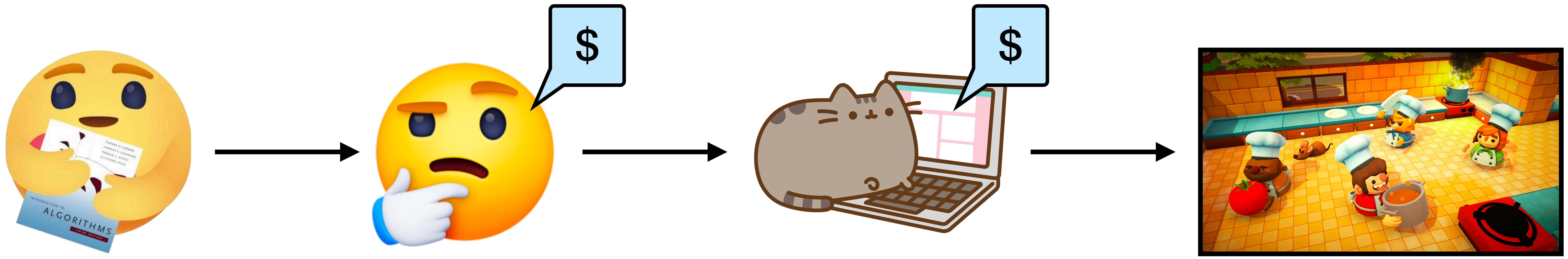


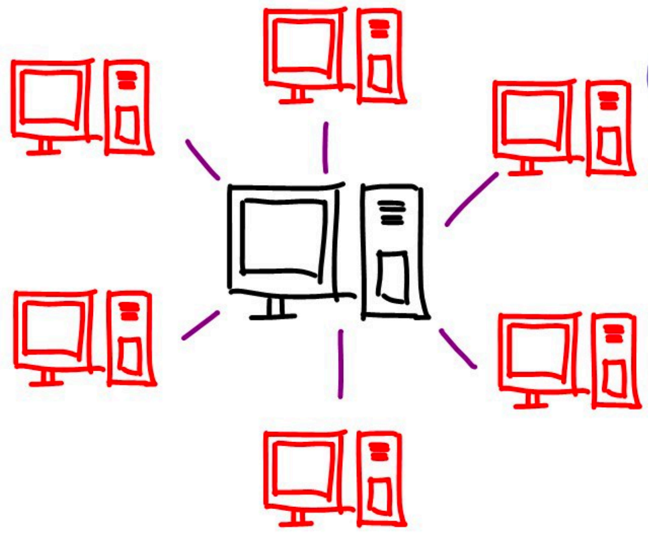
Goal: improve performance with knowledge about locality in the input

Locality-First Algorithm Development on Alternative Computing Platforms

Although this talk demonstrated the potential for the locality-first strategy on multicores, there is significant potential for the approach on **other platforms.**

The Locality-First Strategy





Distributed Systems [Peleg00]

GPUs [Harris13]

Research Vision

My research vision is to make design, analysis, and usage of **parallel and cache-efficient** algorithms and data structures as easy as serial computing in a flat memory.