# When to Apply Parallelism in Practice


Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA


# Learning Objectives in this Part of the Lesson

- Understand the meaning of key concepts associated with parallel programming
- Know when to apply parallelism in practice
  - i.e., what conditions must apply to choose parallelism as the programming paradigm

# When to Apply Parallelism in Practice

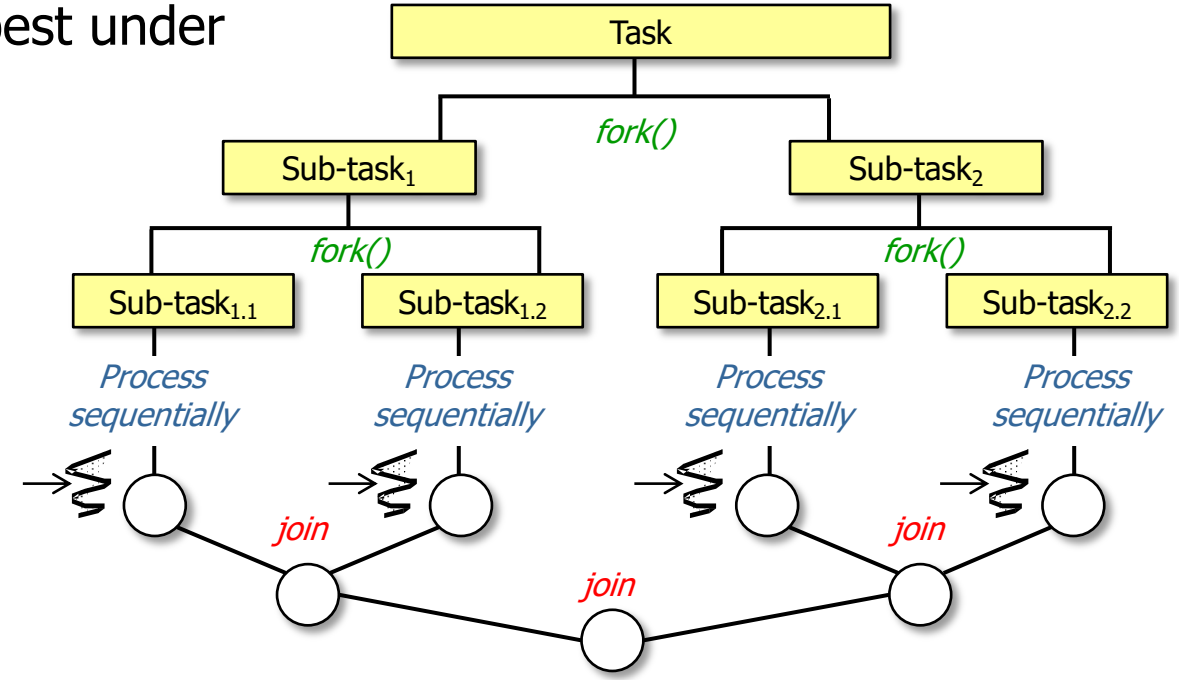
# When to Apply Parallelism in Practice

- Parallelism is not a panacea!!


Particularly when there's contention for shared resources

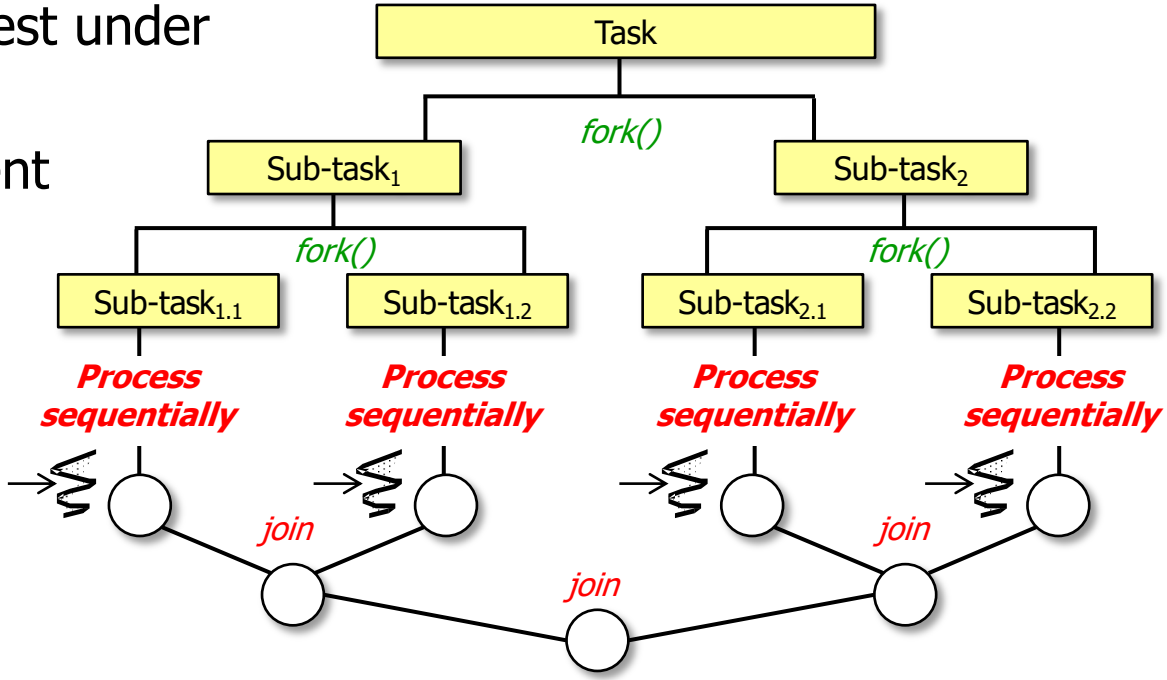
# When to Apply Parallelism in Practice

- Instead, parallelism works best under certain conditions

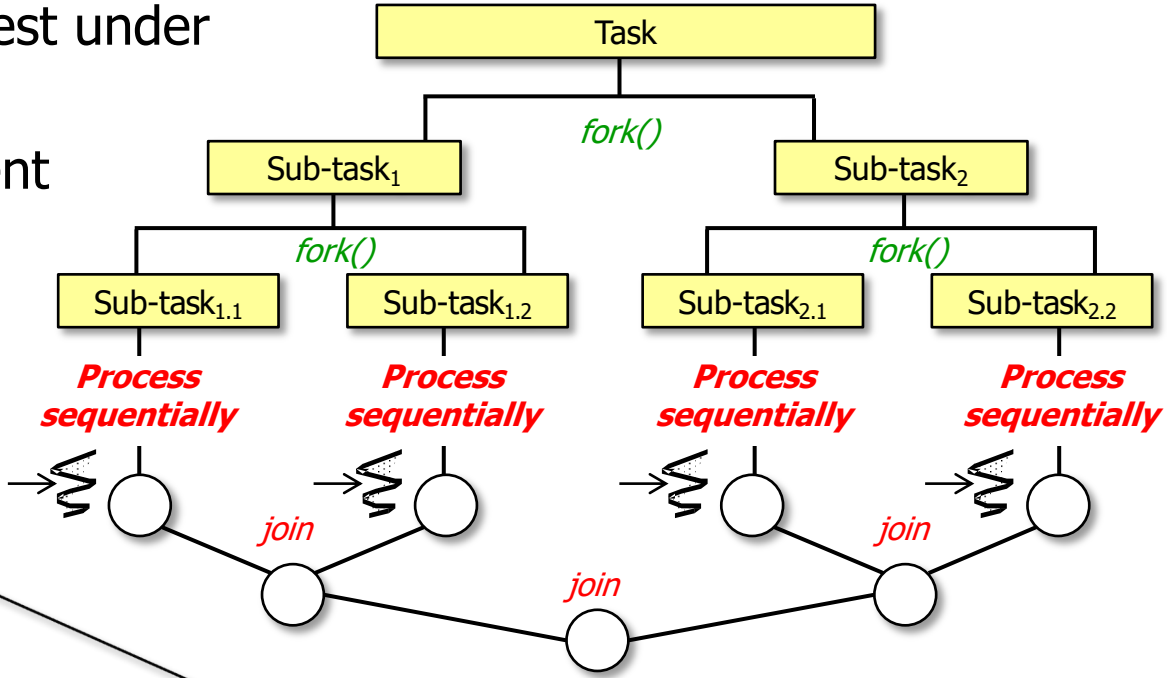
# When to Apply Parallelism in Practice

- Instead, parallelism works best under certain conditions, e.g.
  - When tasks are independent



# When to Apply Parallelism in Practice

- Instead, parallelism works best under certain conditions, e.g.
  - When tasks are independent

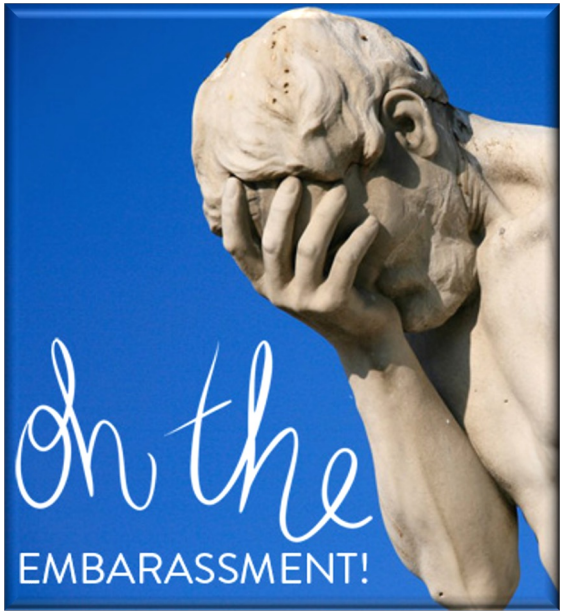


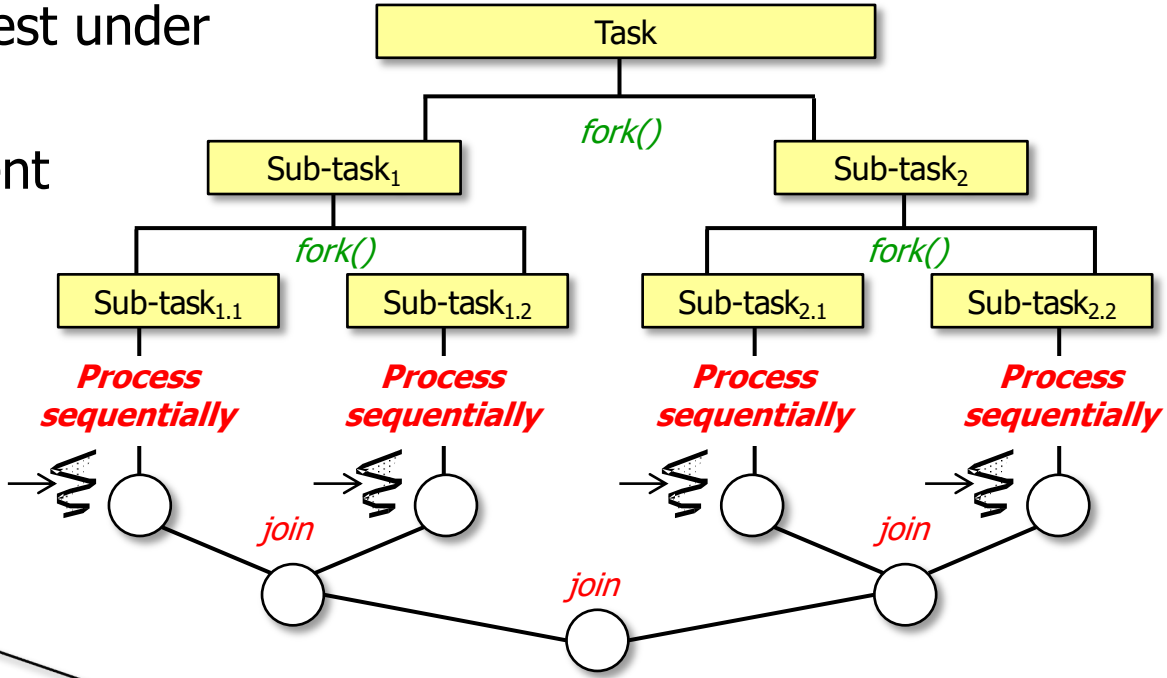

*"Embarrassingly parallel" tasks have little/no dependency or need for communication between tasks or for sharing results between them*

See en.wikipedia.org/wiki/Embarrassingly_parallel

# When to Apply Parallelism in Practice

- Instead, parallelism works best under certain conditions, e.g.
  - When tasks are independent



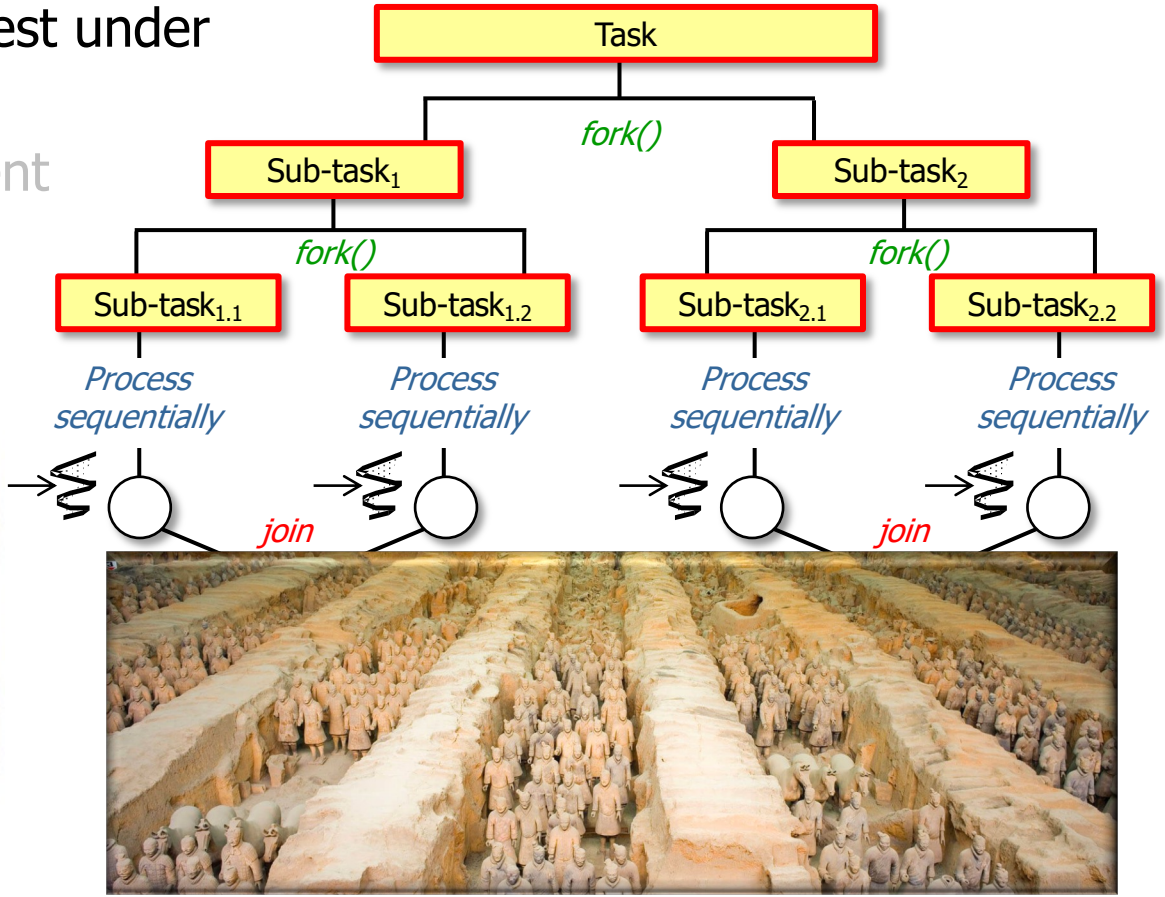




*"Embarrassing" in this context means "over-abundance" or "too much of a good thing"!*

See en.wikipedia.org/wiki/Embarrassment_of_riches

# When to Apply Parallelism in Practice

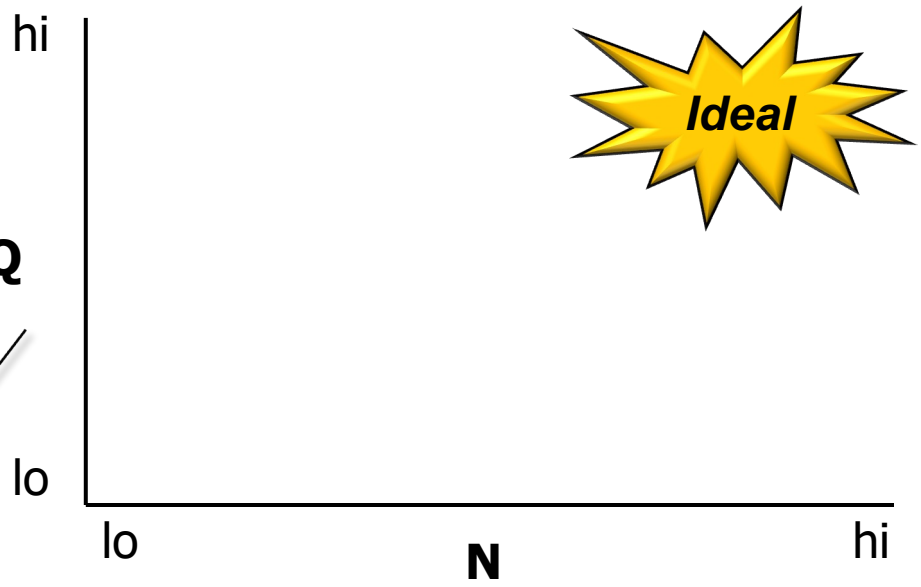
- Instead, parallelism works best under certain conditions, e.g.
  - When tasks are independent
  - When there's lots of data & processing to perform




See en.wikipedia.org/wiki/Terracotta_Army

# When to Apply Parallelism in Practice

- Instead, parallelism works best under certain conditions, e.g.
  - When tasks are independent
  - When there's lots of data & processing to perform
    - The "N*Q" heuristic estimates the benefit of parallelism

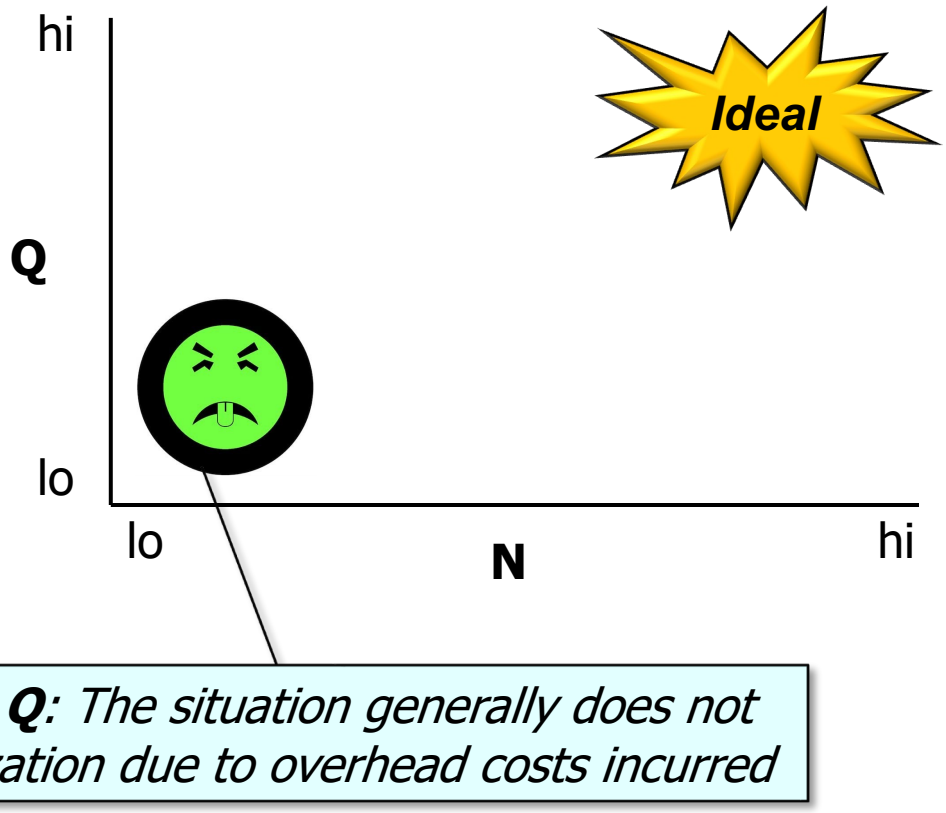



- *N is the # of data elements to process*
- *Q quantifies CPU processing intensity for each data element*

See on-sw-integration.epischel.de/2016/08/05/parallel-stream-processing-with-java-8-stream-api

# When to Apply Parallelism in Practice

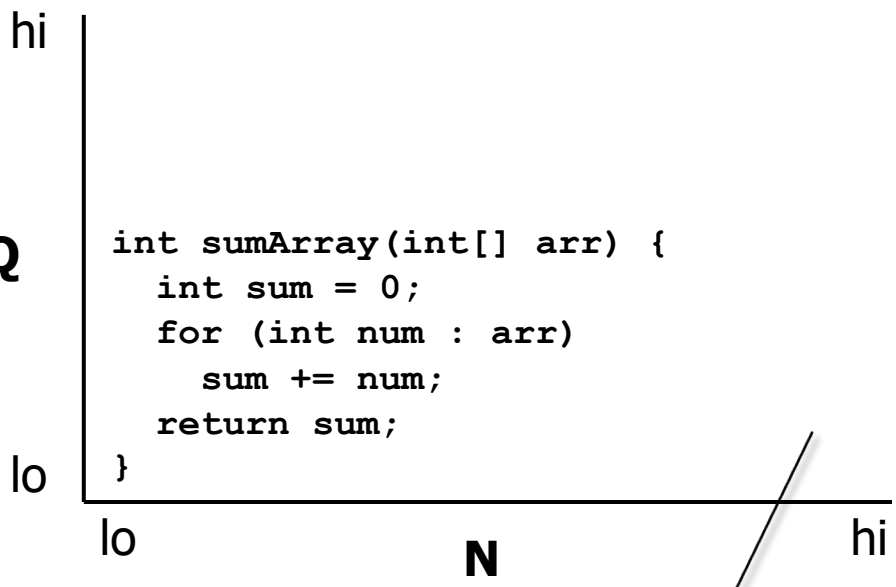
- Instead, parallelism works best under certain conditions, e.g.
  - When tasks are independent
  - When there's lots of data & processing to perform
    - The "N*Q" heuristic estimates the benefit of parallelism




In this case, it's usually best to stick with sequential programming

# When to Apply Parallelism in Practice

- Instead, parallelism works best under certain conditions, e.g.
  - When tasks are independent
  - When there's lots of data & processing to perform
    - The "N*Q" heuristic estimates the benefit of parallelism



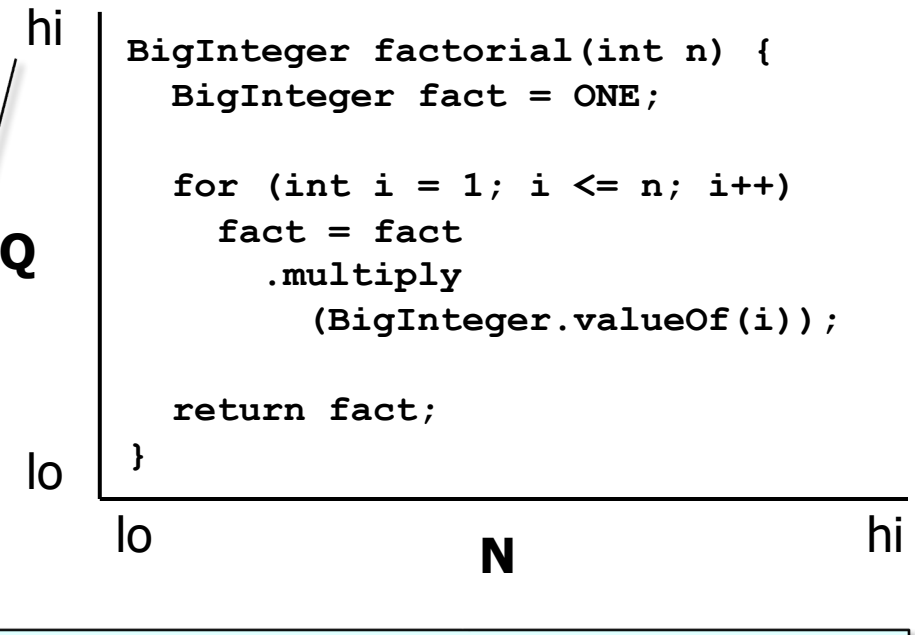

```
int sumArray(int[] arr) {
  int sum = 0;
  for (int num : arr)
    sum += num;
  return sum;
}
```

***High N, Low Q**: The overhead of parallelizing may outweigh the benefits, as the computational work per element is trivial*

Brian Goetz recommends 'N' be > 10,000

# When to Apply Parallelism in Practice

- Instead, parallelism works best under certain conditions, e.g.
  - When tasks are independent
  - When there's lots of data & processing to perform
    - The "N*Q" heuristic estimates the benefit of parallelism



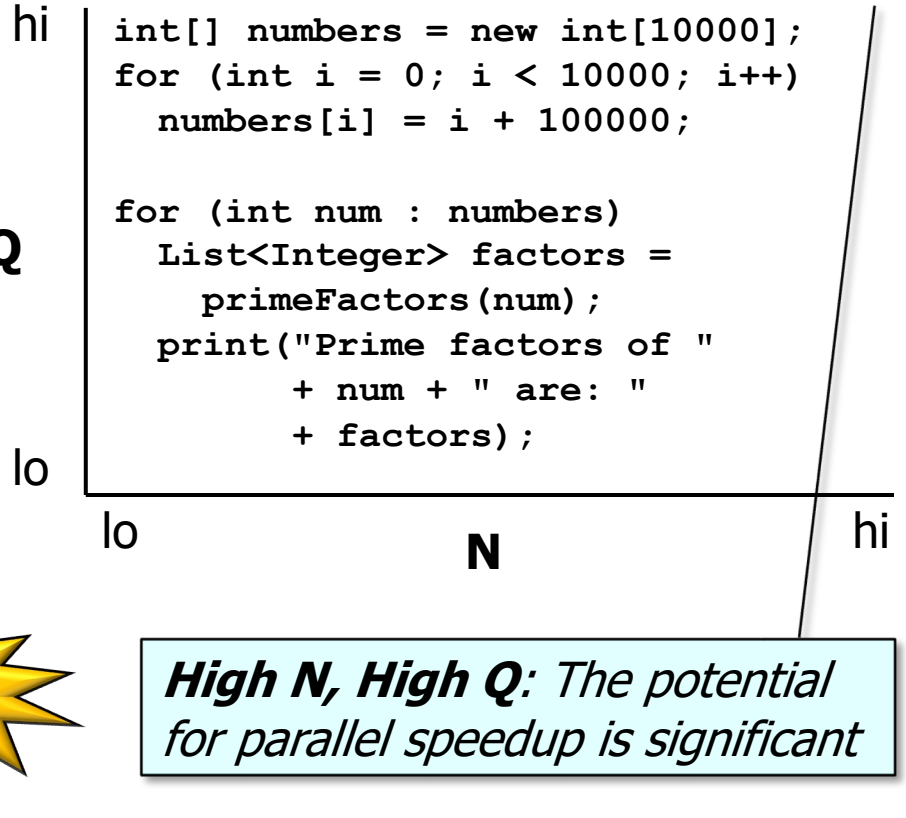

```
BigInteger factorial(int n) {
  BigInteger fact = ONE;

  for (int i = 1; i <= n; i++)
    fact = fact
      .multiply
        (BigInteger.valueOf(i));

  return fact;
}
```

***Low N, High Q**: The computational workload for each data element is high, so even a small N can benefit from parallelization because work can be partitioned across multiple cores, thereby reducing the total time for computation*

Often seen in simulations, complex math computations, or graphics rendering

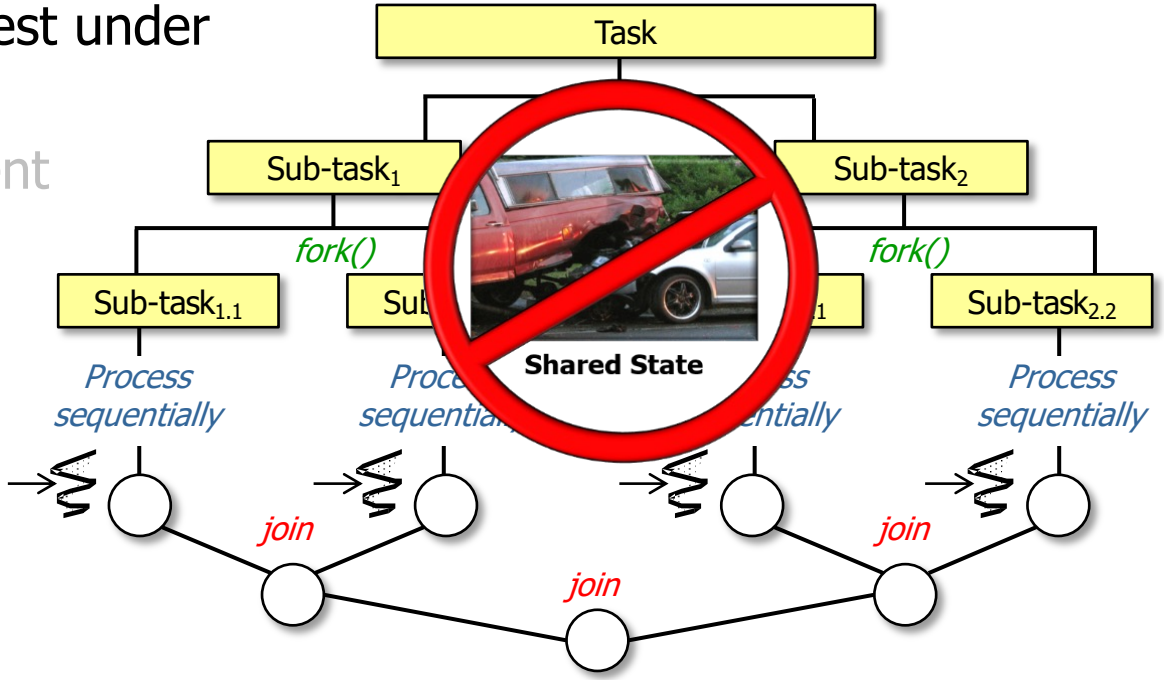
# When to Apply Parallelism in Practice

- Instead, parallelism works best under certain conditions, e.g.
  - When tasks are independent
  - When there's lots of data & processing to perform
    - The "N*Q" heuristic estimates the benefit of parallelism



# When to Apply Parallelism in Practice

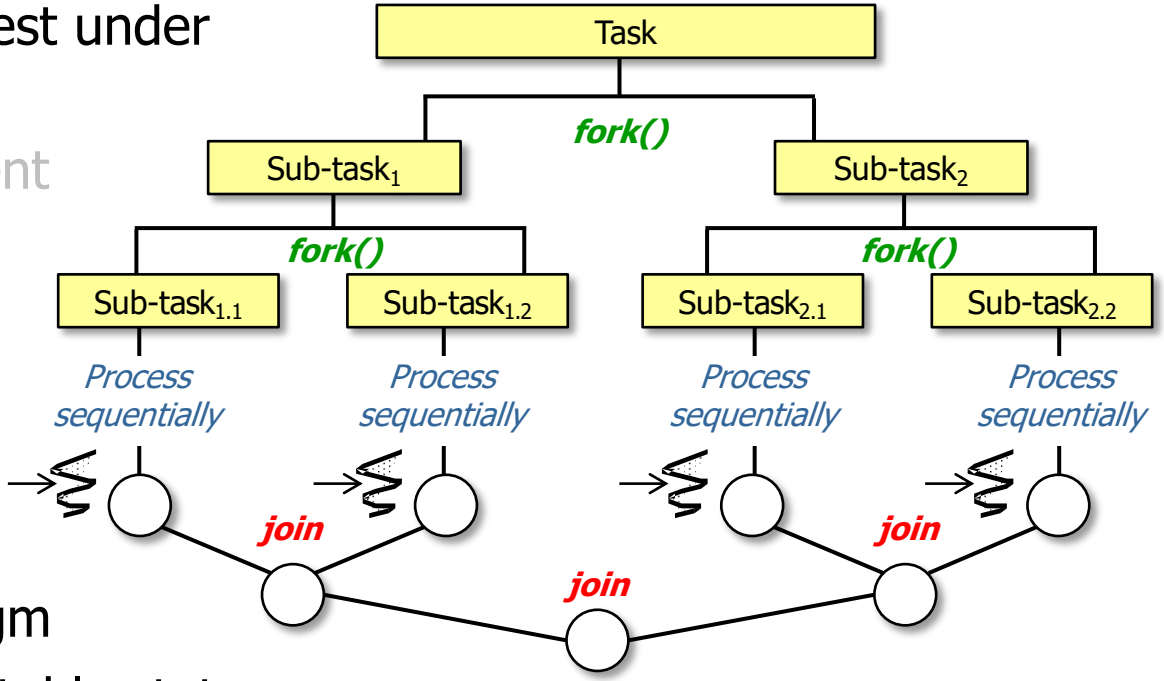
- Instead, parallelism works best under certain conditions, e.g.
  - When tasks are independent
  - When there's lots of data & processing to perform
  - When tasks neither block nor share mutable state




See henrikeichenhardt.blogspot.com/2013/06/why-shared-mutable-state-is-root-of-all.html

# When to Apply Parallelism in Practice

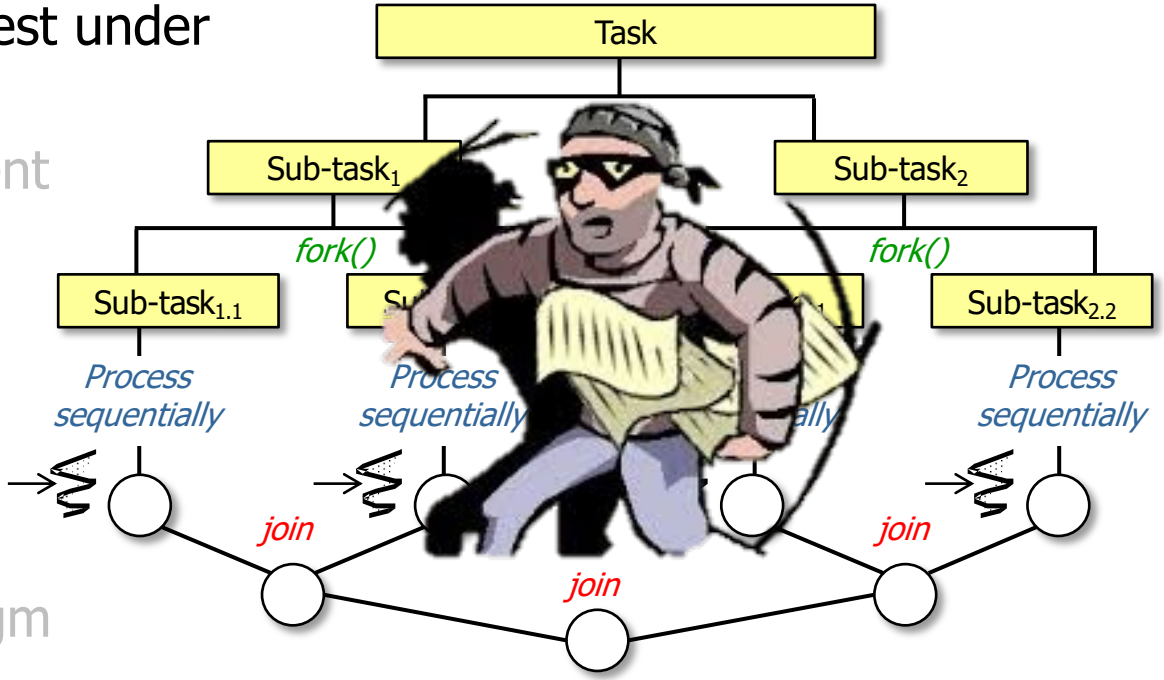
- Instead, parallelism works best under certain conditions, e.g.
  - When tasks are independent
  - When there's lots of data & processing to perform
  - When tasks neither block nor share mutable state
    - Hence Java's focus on
      - The "fork-join" paradigm
        - To avoid sharing mutable state




See en.wikipedia.org/wiki/Fork-join_model

# When to Apply Parallelism in Practice

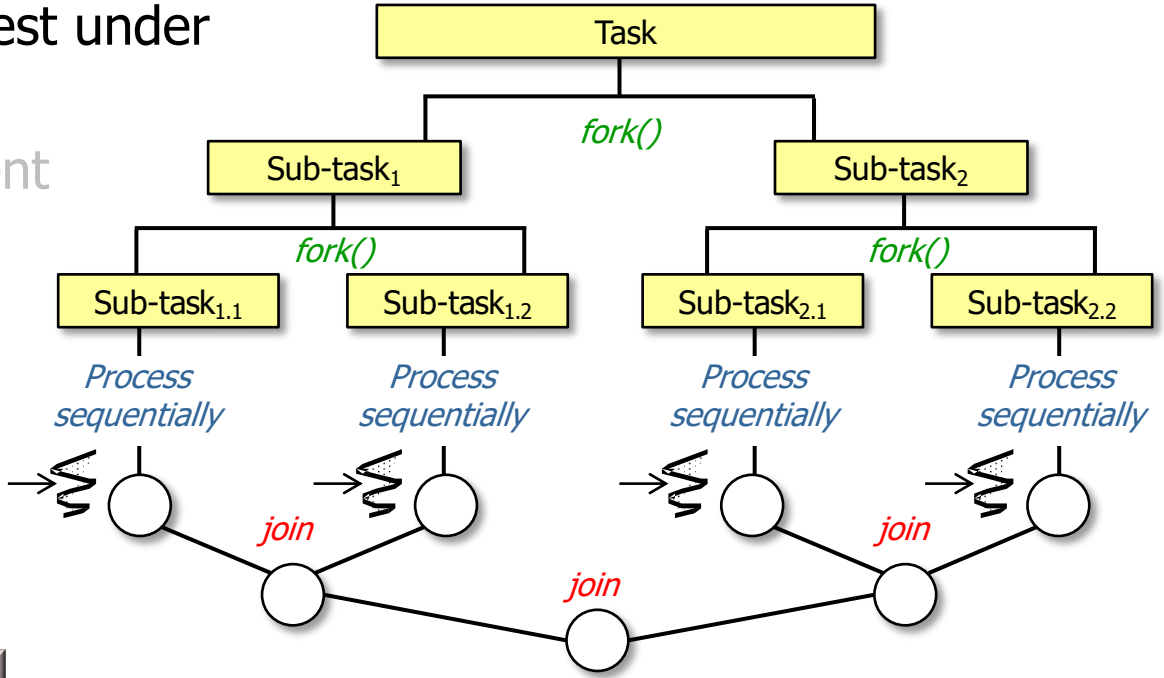
- Instead, parallelism works best under certain conditions, e.g.
  - When tasks are independent
  - When there's lots of data & processing to perform
  - When tasks neither block nor share mutable state
    - Hence Java's focus on
      - The "fork-join" paradigm
      - "Work-stealing"
        - To avoid blocking




See en.wikipedia.org/wiki/Work_stealing

# When to Apply Parallelism in Practice

- Instead, parallelism works best under certain conditions, e.g.
  - When tasks are independent
  - When there's lots of data & processing to perform
  - When tasks neither block nor share mutable state
  - When there are many cores and/or processors





See en.wikipedia.org/wiki/Multi-core_processor & en.wikipedia.org/wiki/Multiprocessing

# End of When to Apply Parallel Programming in Practice