ExaBiome: Exascale Solutions to Microbiome Analysis

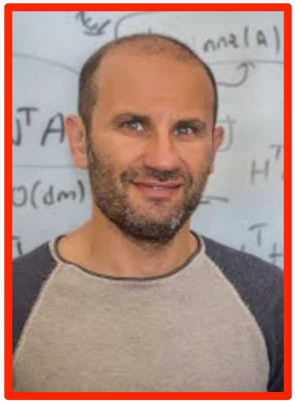
https://sites.google.com/lbl.gov/exabiome/

Kathy Yelick (PI)
LBNL, LANL, and JGI

January 2017

The Team from LBNL, LANL and JGI


Kathy Yelick


Lenny Oliker


Dan Rokhsar


Aydin Buluc


Patrick Chain


Nikos Kyrpides


Steve Hofmeyr


Rob Egan


Natalia Ivanova


Ariful Azad

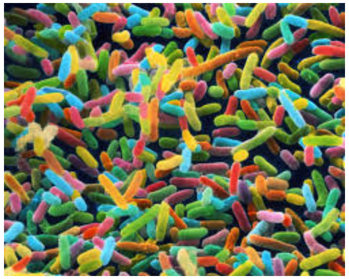

Pavel Senin

Evangelos Georganas

ExaBiome: Exascale Solutions for Microbiome Analysis

- **Microbes:** single cell organisms, such as bacteria and viruses
- **Microbiomes:** communities of 1000s of microbial species, less than 1% individually culturable in a lab (and thus sequenced)
- **Metagenomics:** genome sequencing on these communities (growing exponentially)

Environment



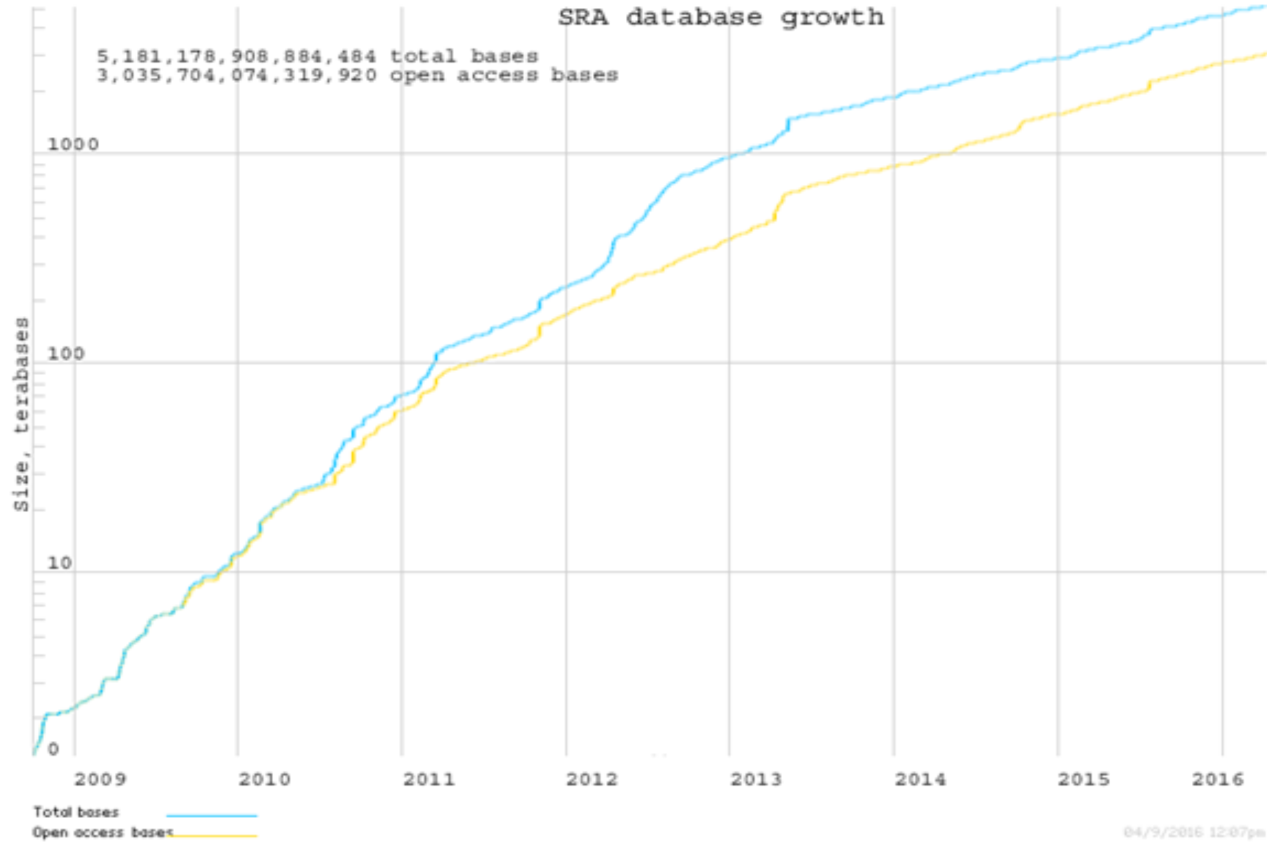
(Health)

Bio-Energy

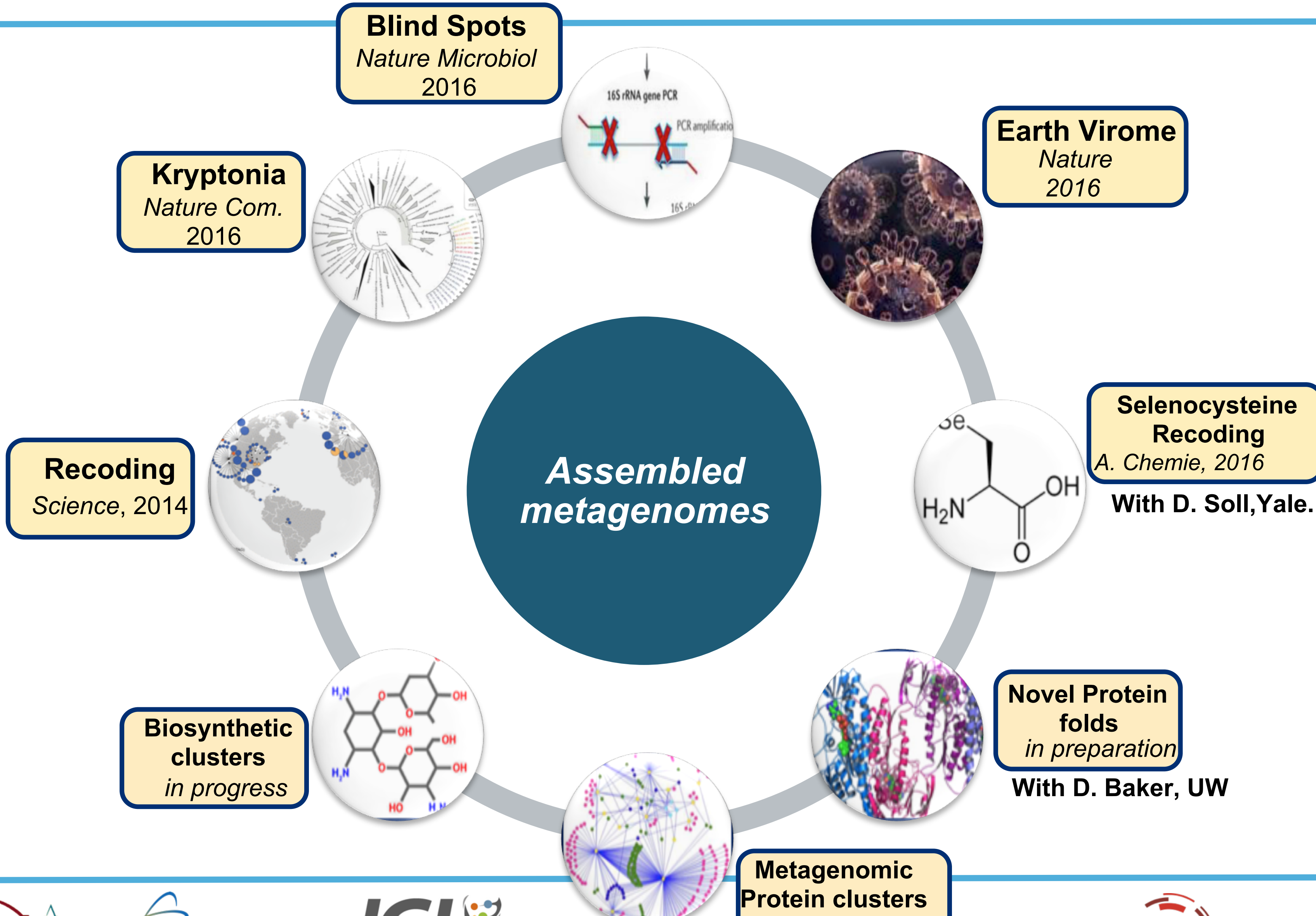
Bio-Manufacturing

Metagenome database growth

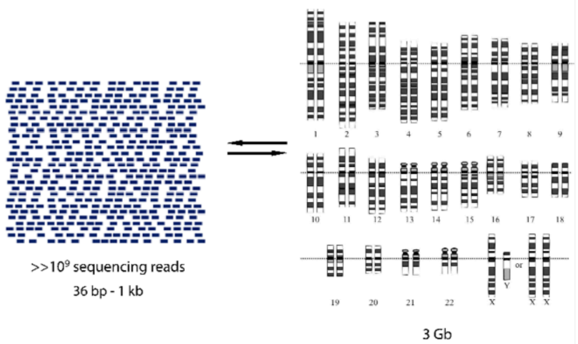
- **Metagenomic data double every 11 months**

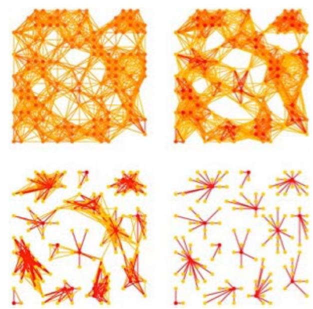


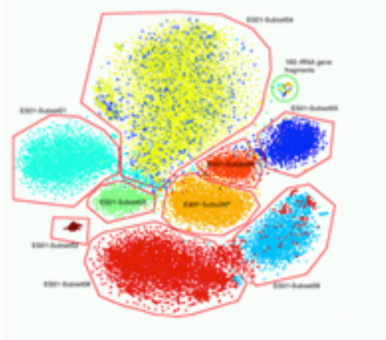
Metagenomics data mining efforts at JGI



ExaBiome: Exascale Solutions for Microbiome Analysis







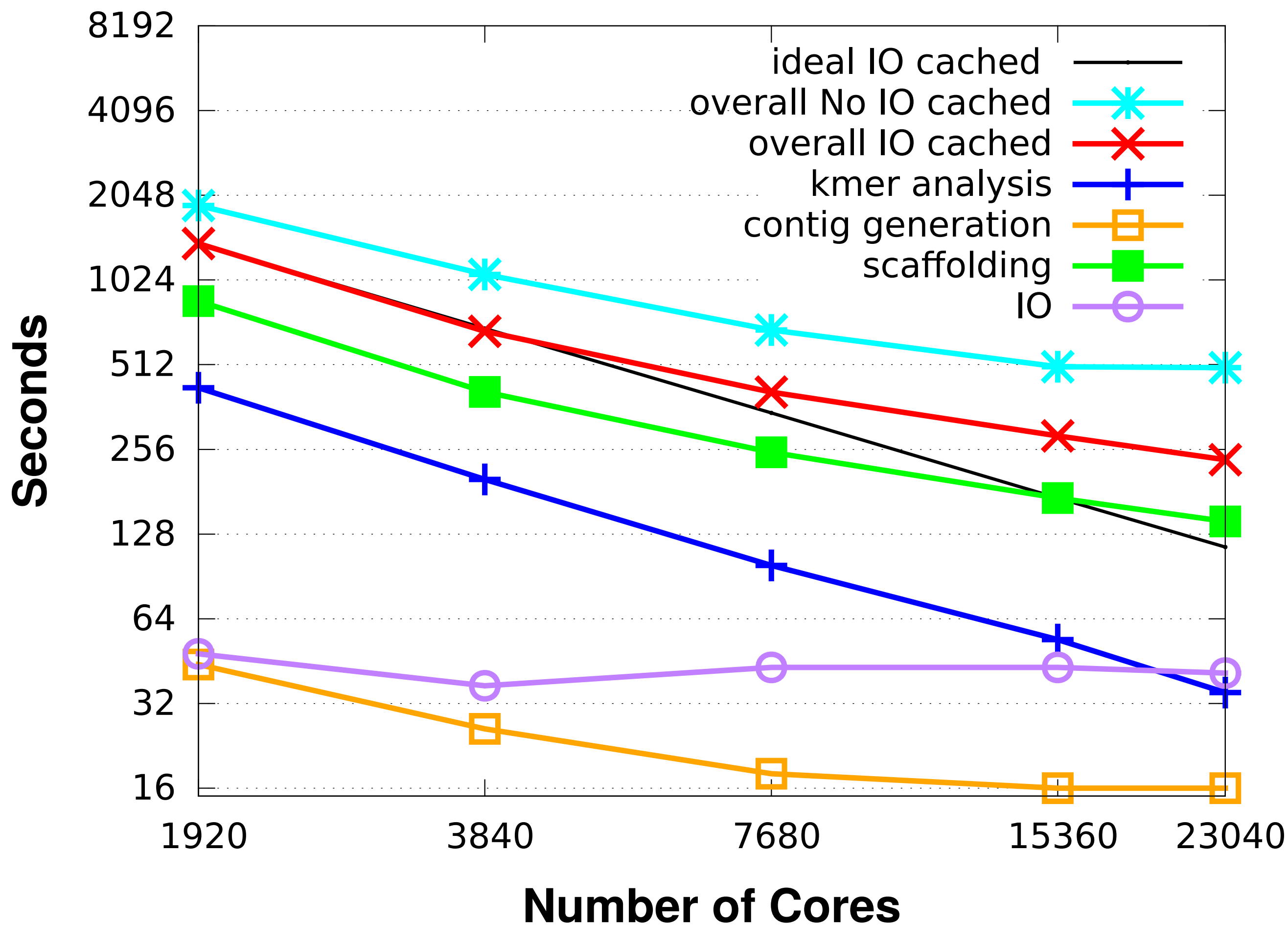
Metagenome Assembly	Protein Clustering	Comparative Analysis
Graph algorithms, Hash Tables, alignment (Smith-Waterman)	Machine learning (clustering), sparse linear algebra / graphs	Alignment, Machine learning (dimensionality reduction), linear algebra
Fine-grained comm., all-to-all, remote atomics and fast I/O	Fast barriers, subset reductions	All-to-all

PI: Katherine Yelick (LBNL)

Metagenome assembly is hard

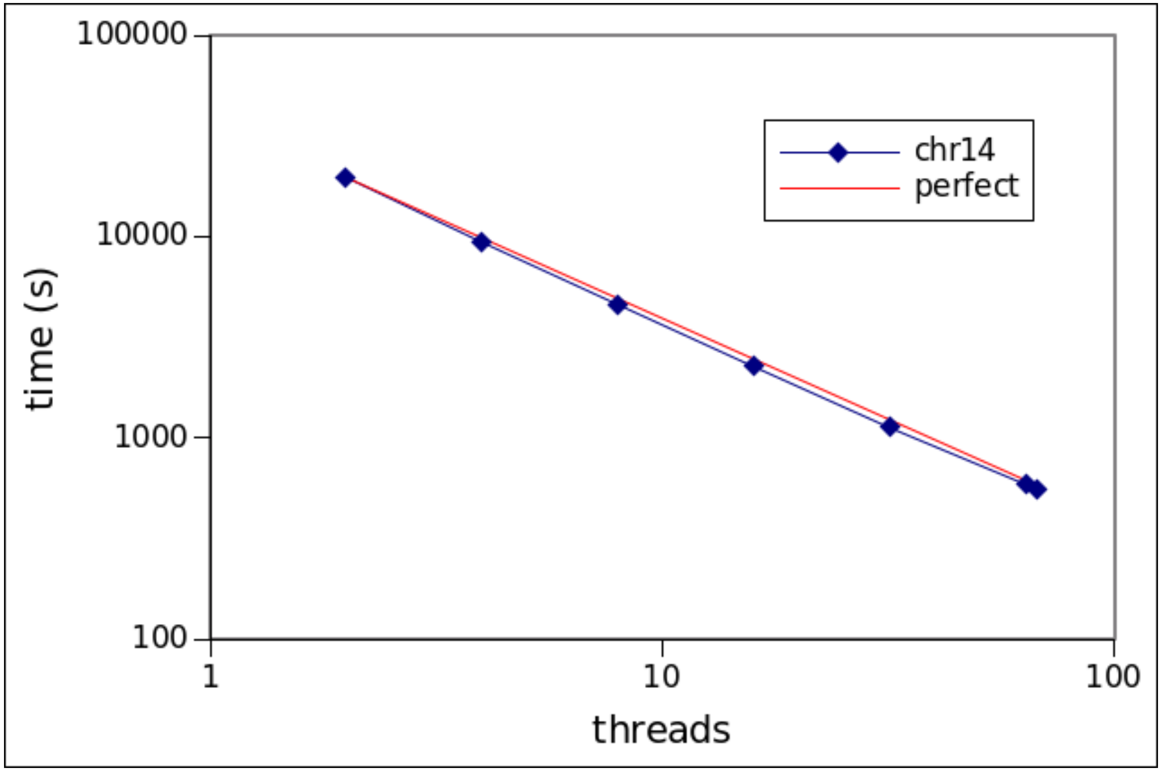
- **There is no genome reference.**
- **Reads are significantly shorter than whole genome.**
  - –Reads consist of 20 to 30K bases
  - –Genomes vary in length and complexity – up to 30G bases
- **Reads include errors.**
- **Genomes have repetitive regions.**
  - –Repetitive regions increase genome complexity
- **Microbial genomes occur with thousands of others**
  - –The coverage (frequency) may vary be orders of magnitude

Strong scaling (human genome) on Cray XC30



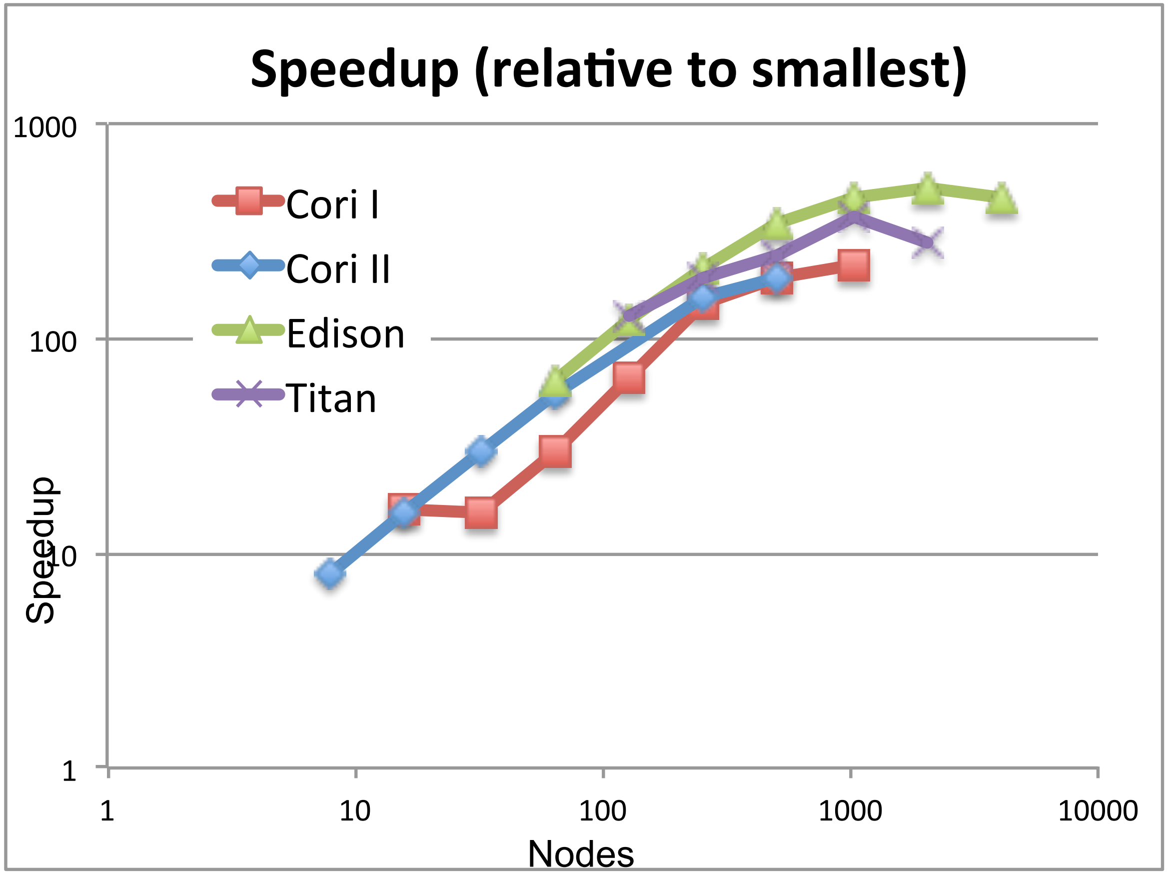
Single Node Scaling on KNL

Strong Scaling (small dataset, chromosome14)



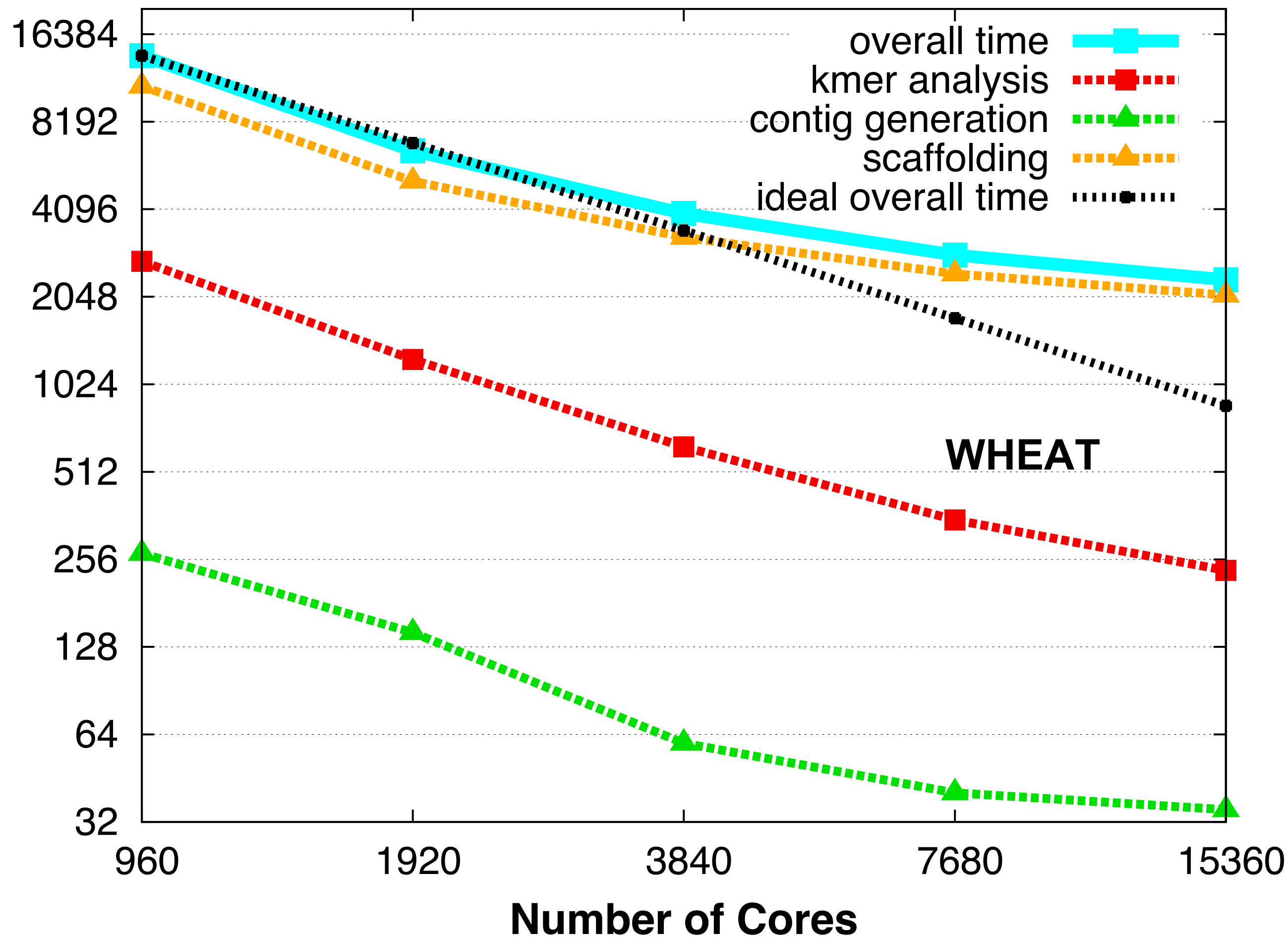
UPC (with MPI in K-mer analysis)

Speedup on Human Data (single genome)



- Complete assembly of human genome in **4 minutes using 23K cores.**
- **700x speedup over** original Meraculous code used in production (only ran on shared memory where it took **2,880 minutes**)

Strong scaling (wheat genome) on Cray XC30



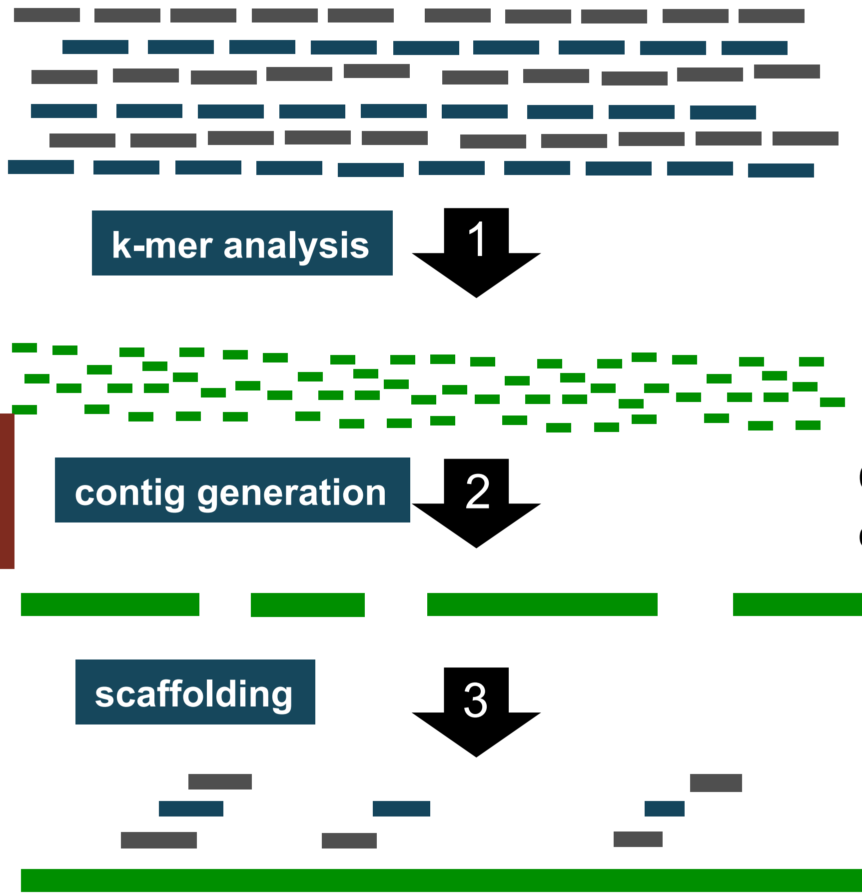
Early version of HipMer were used for the first whole genome assembly of wheat

Original HipMer Pipeline Summary (Single Genome)

reads

Input: Reads that may contain errors

Chop reads into k-mers, process k-mers to **exclude errors**



***k*-mers**

NERSC-9 “Meraculous” benchmark

Construct & traverse de Bruijn graph of k-mers, generate contigs

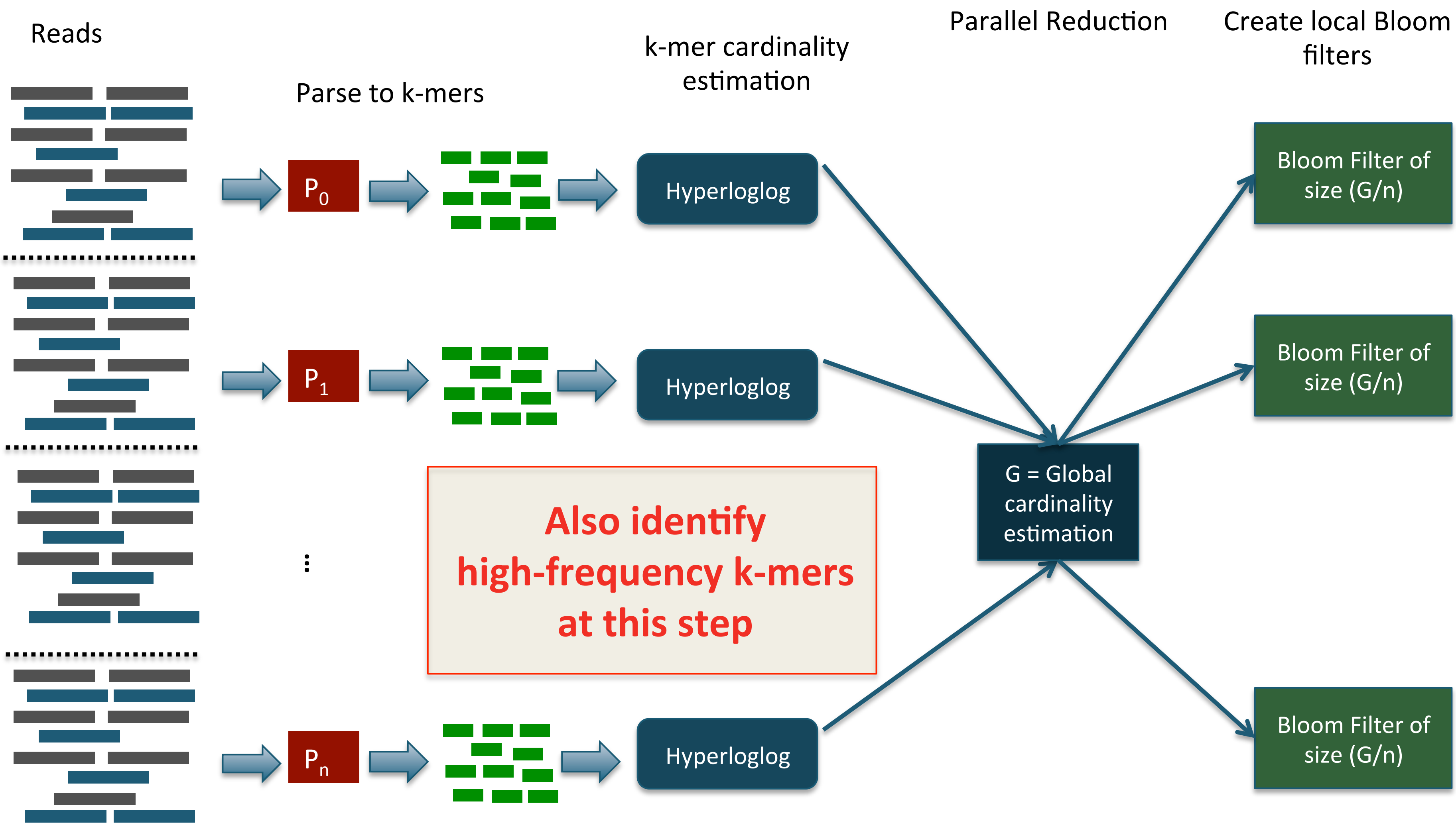
contigs

Leverage read information to link contigs and generate scaffolds.

scaffolds

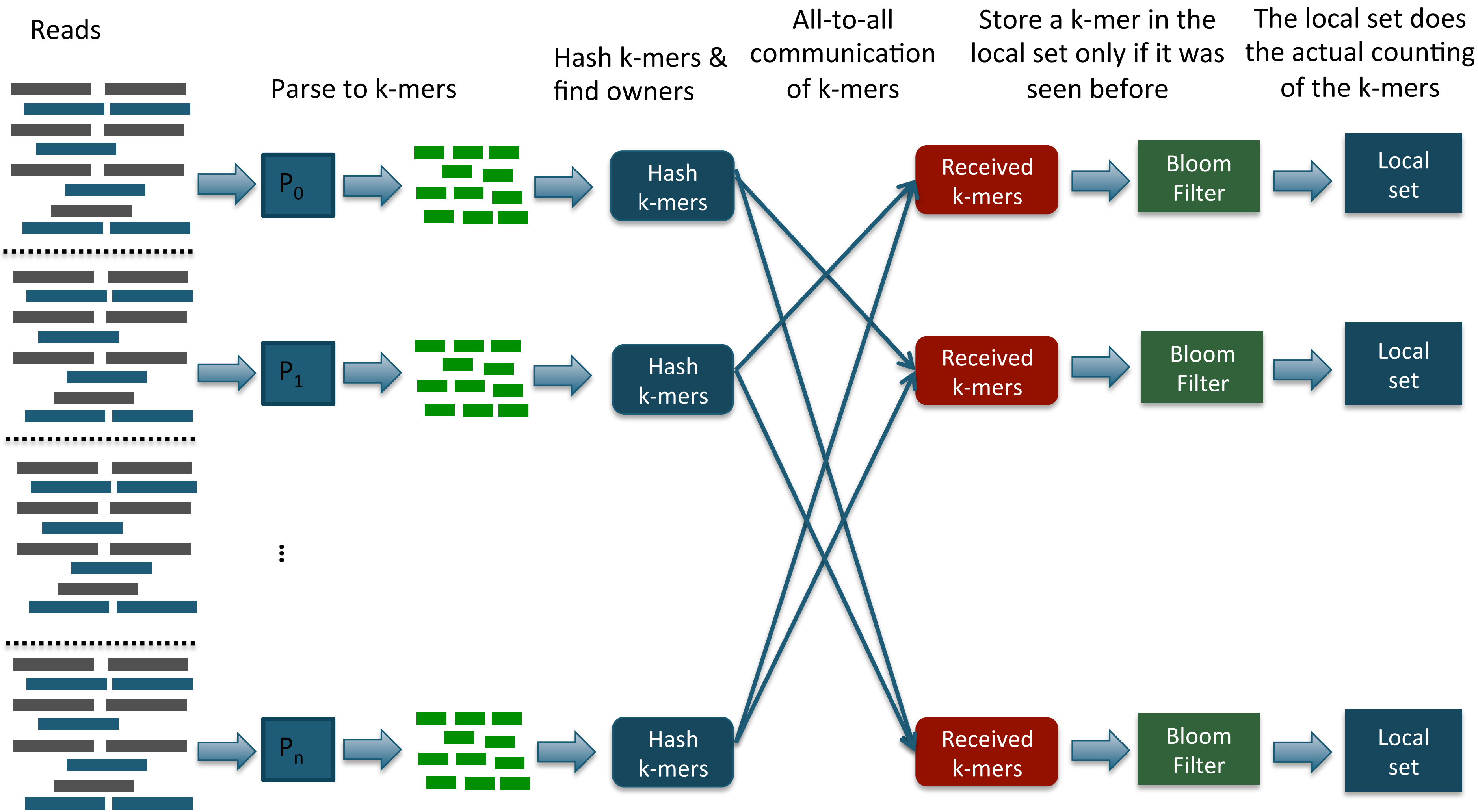
(Assembly output is generally called “contigs” even when a scaffolding phase is included.)

K-Mer Analysis: Pass 1 (I/O + Independent + Reduce)



Also identify high-frequency k-mers at this step

K-Mer Analysis: Pass 2 (Iterative All-to-All)



Bloom Filter

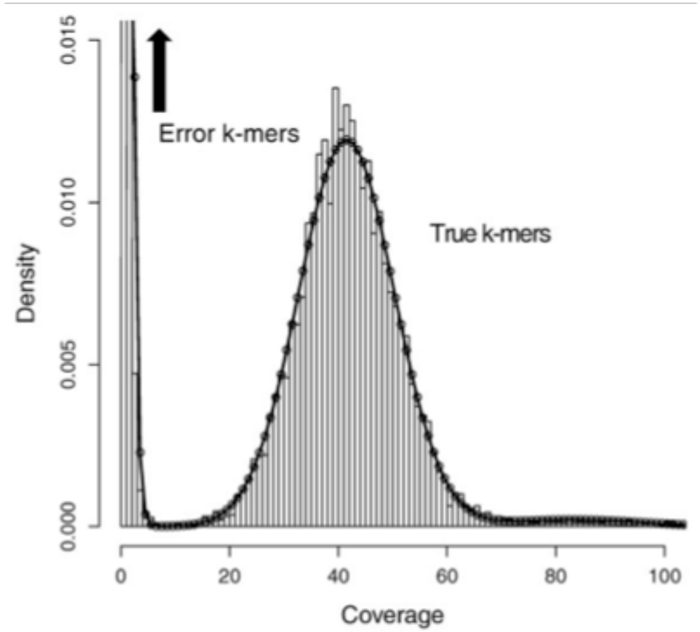
Bloom filter is a *probabilistic* data structure used for membership queries

- **Given a bloom filter, we can ask:**

"Have we seen this k-mer before?"

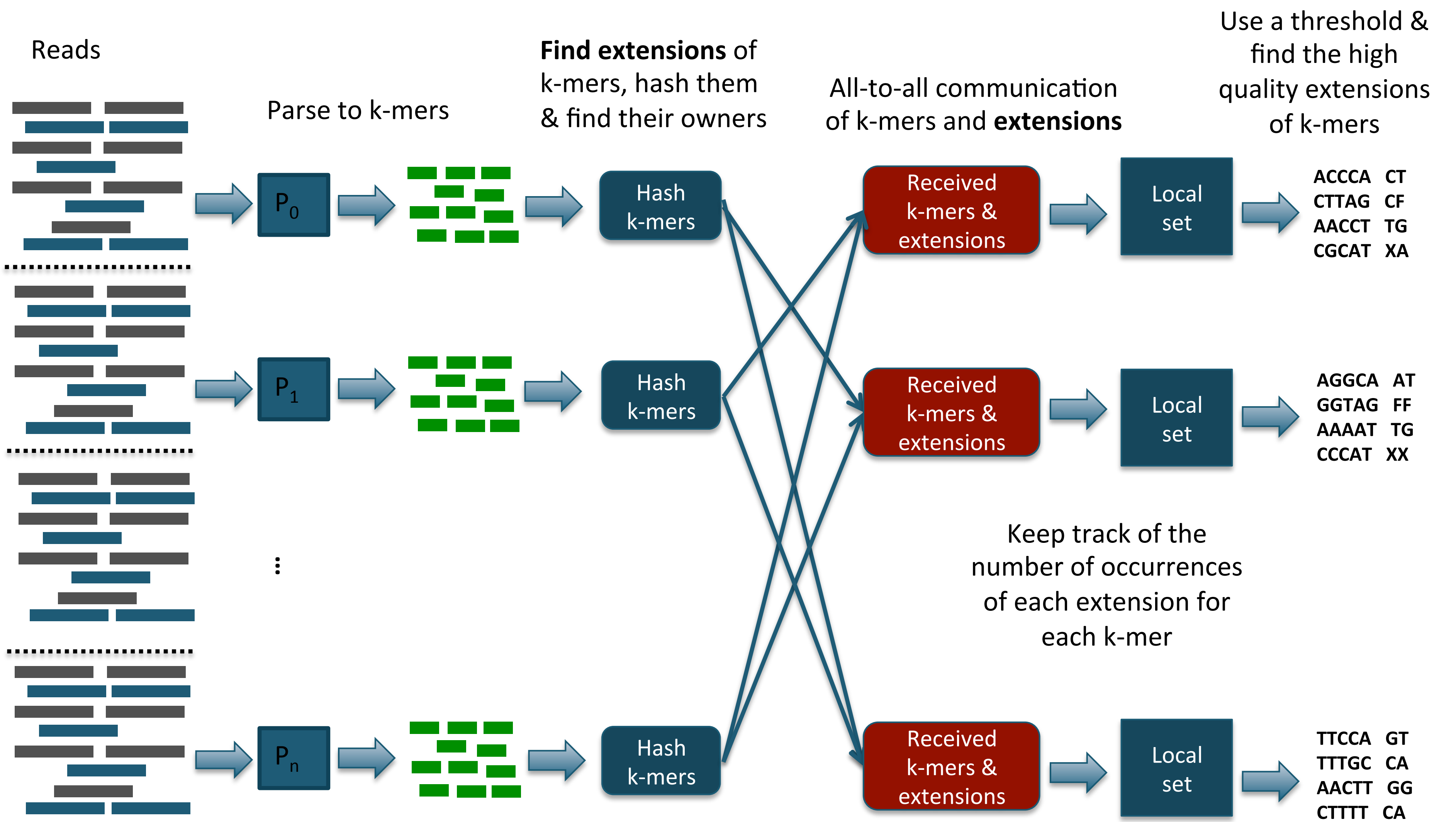
- **No false negatives.**
- **May have false positives**

(in practice 5% false positive rate)



k-mers with frequency =1 are useless (either error or can not be distinguished from error), and can safely be eliminated.

K-Mer Analysis: Step 3 (All-to-All)

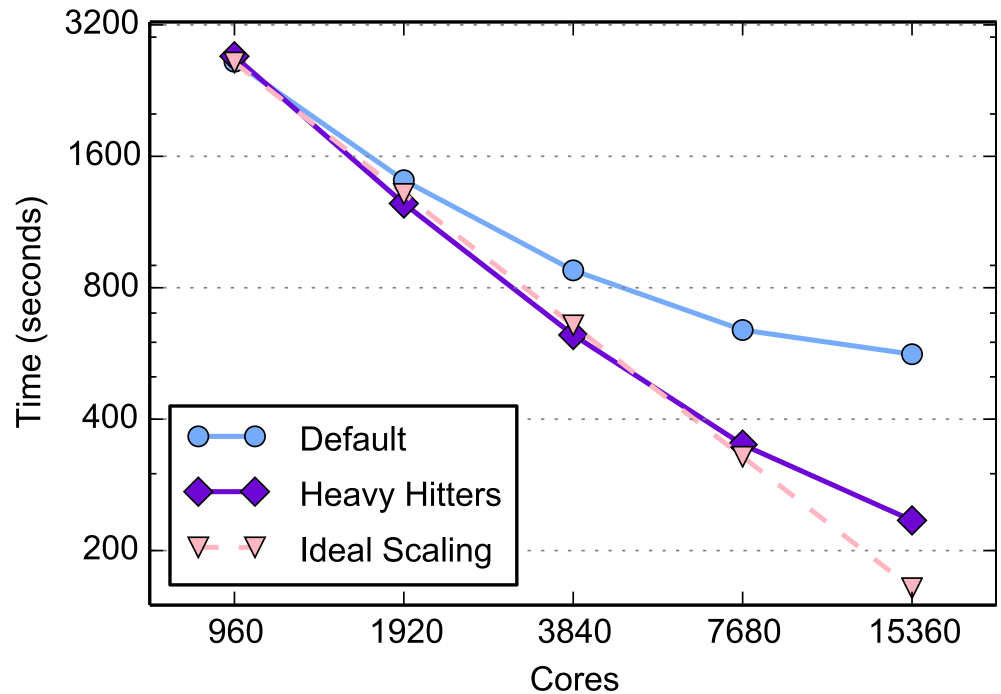


Heavy Hitters

Long-tailed distribution for genomes with repetitive content:

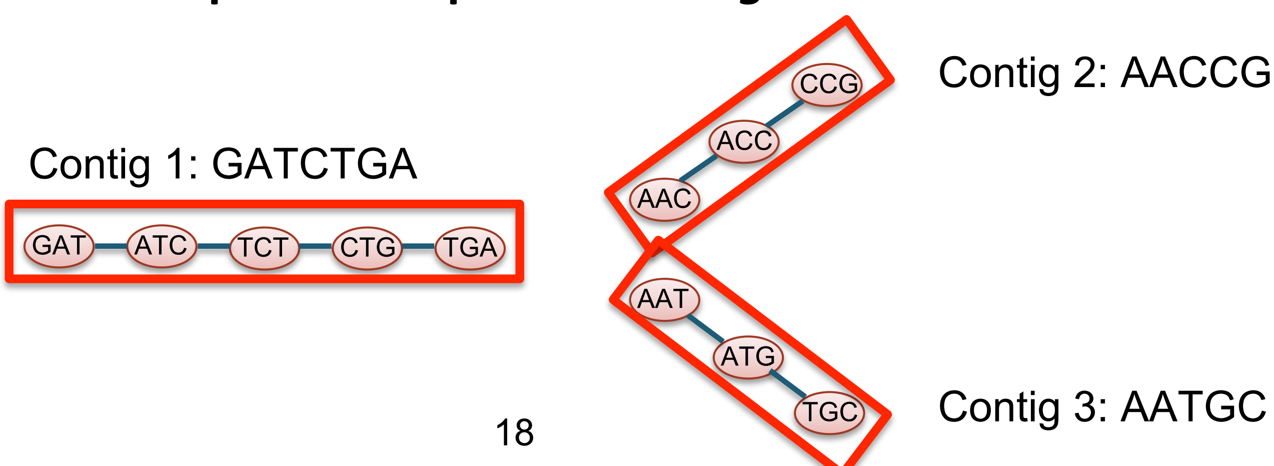
- The maximum count for any k-mer in the wheat dataset is **451 million**
- Our original scheme (SC'14) was "owner counts", after an all-to-all
- Counting an item w/ 451 million occurrences alone is **load imbalanced**

Solution: Quickly identify high-frequency k-mers using minimal communication during the "cardinality estimation" step and treat them specially by using local counters.



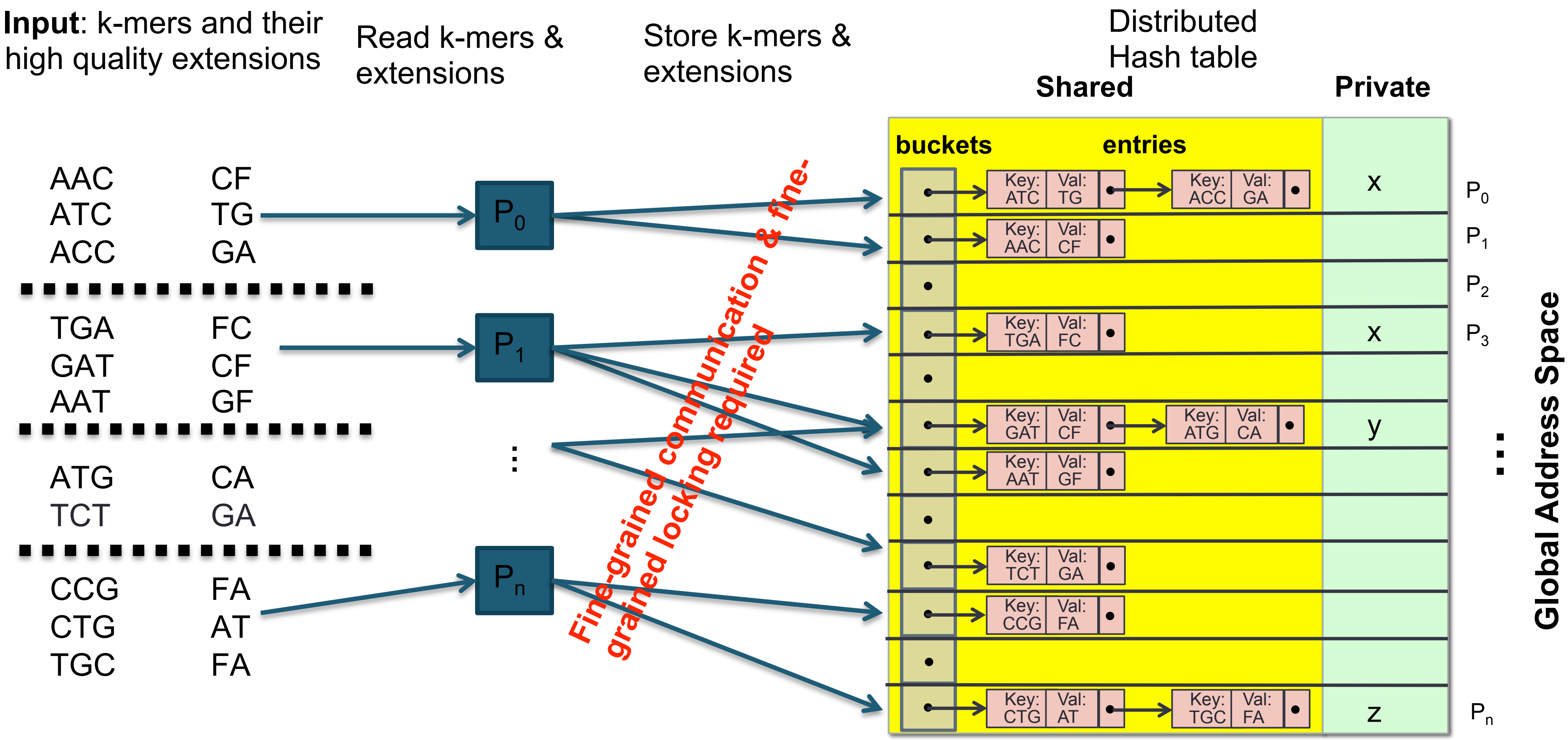
Distributed De Bruijn Graph

- **The de Bruijn graph of k-mers is represented as a hash table.**
- **A k-mer is a node in a graph ⇔ a k-mer is an entry (key) in the hash table.**
- **An edge in the graph connects two nodes that overlap in k-1 bases.**
- **The edges in the hash table can be stored efficiently by storing the extensions of the k-mers as their corresponding values.**
- **The connected components represent *contigs*.**



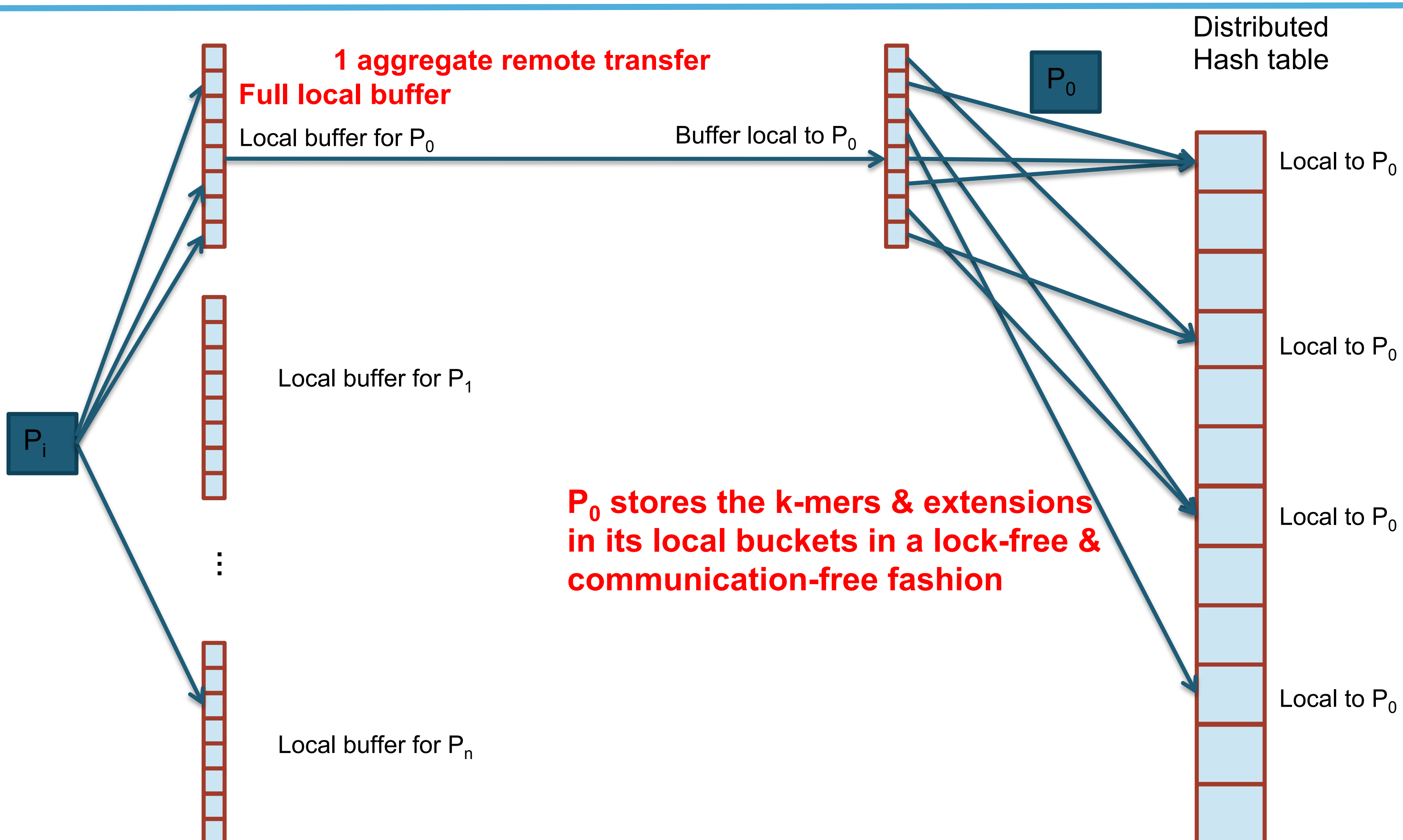
Parallel De Bruijn Graph Construction

Graph construction, traversal, and all later stages are written in UPC to take advantage of its global address space



BERKELEY LAB · Los Alamos NATIONAL LABORATORY EST.1943 · JGI JOINT GENOME INSTITUTE · ECP EXASCALE COMPUTING PROJECT

Aggregating Stores Optimization



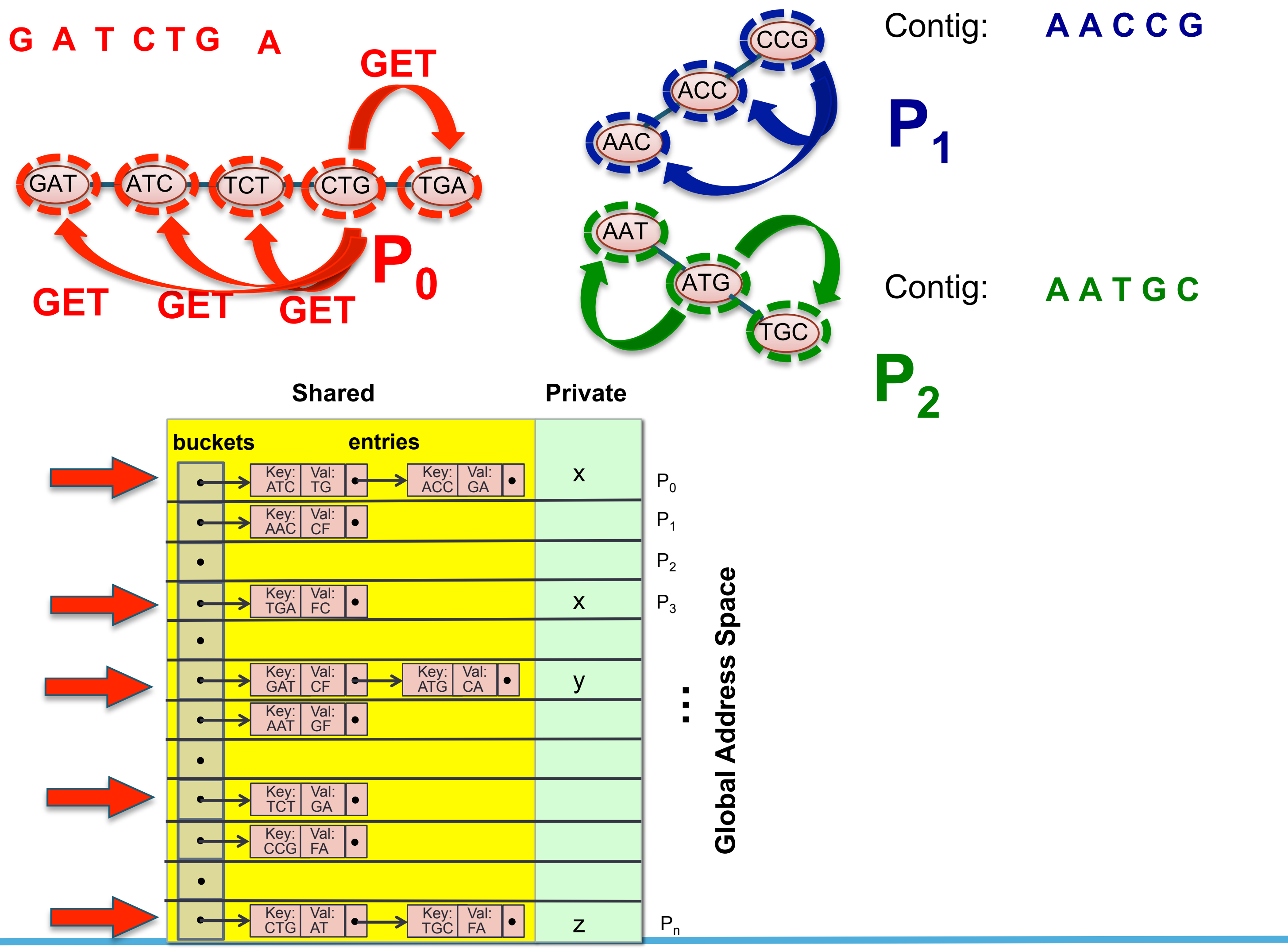
Parallel Graph Construction Challenge

- **Challenge 1: Hash table for de Bruijn graph is huge (at least multiple terabytes)**
  - **Solution:** Distribute the graph over multiple processors.

- **Challenge 2: Parallel hash table construction introduces communication and synchronization costs**
  - **Solution:** Split the construction in two phases and aggregate messages → **10x-20x performance improvement.**

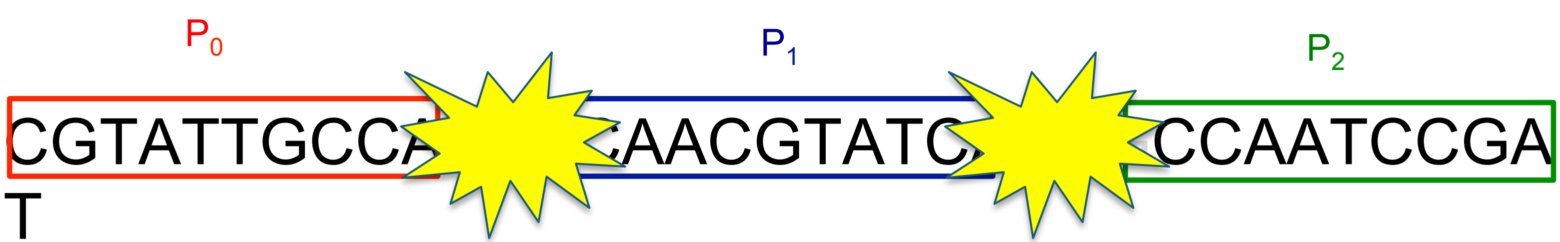
Parallel De Bruijn Graph Traversal

Algorithm: Pick a random k-mer and expand connected component by consecutive lookups in the distributed hash table.

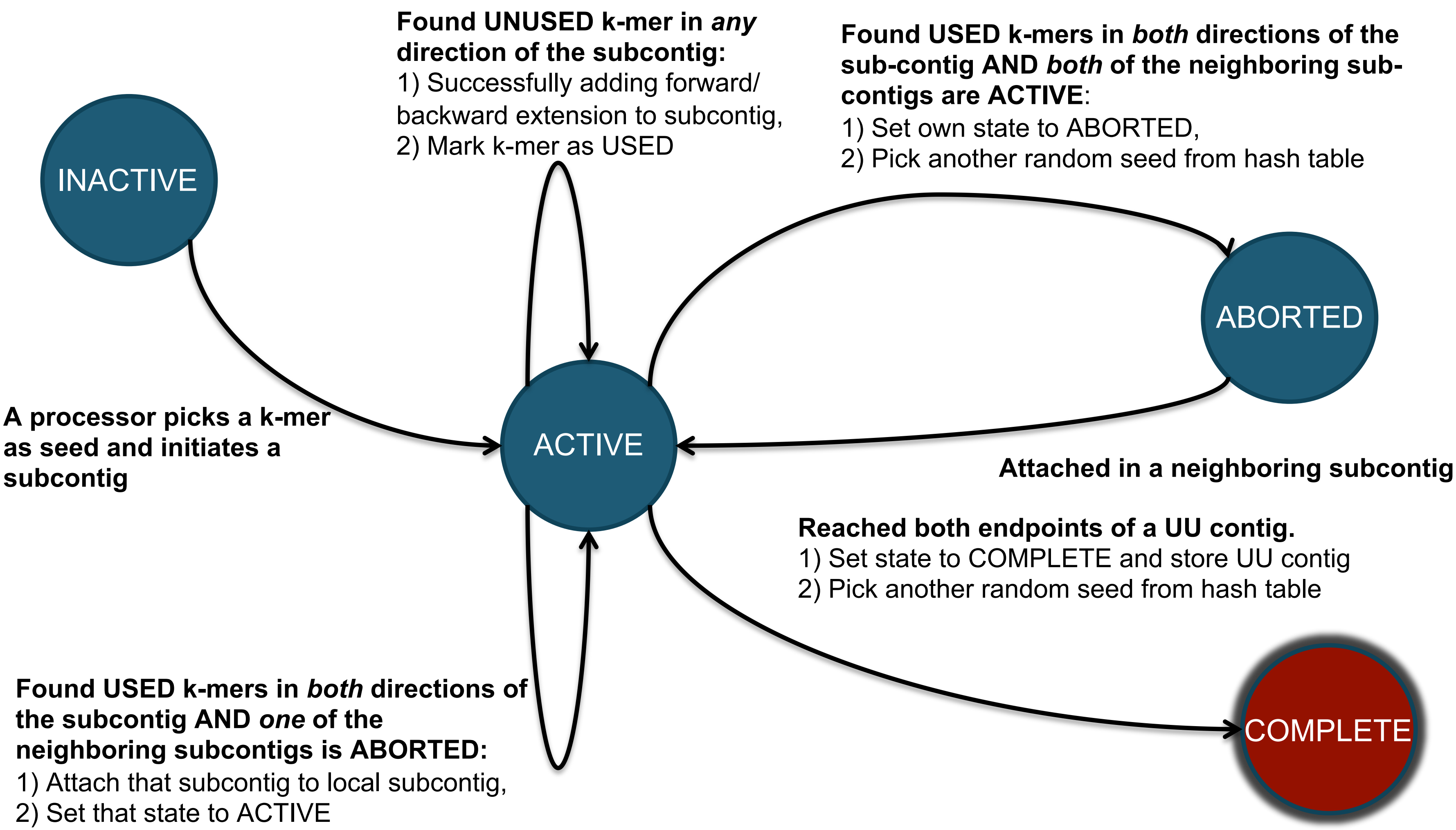


Parallel De Bruijn Graph Traversal

- **<u>Algorithm:</u> Pick a random k-mer and expand connected component by consecutive lookups in the distributed hash table.**
- **Fine-grained, irregular, remote accesses. Need fine-grained parallelization.**
  - Worst case: the result is a single very long chain (high-diameter graph).
  - Global address space and one-sided communication simplifies logic.
  - If multiple processors are working on the same connected component, they cooperate via a **lightweight synchronization protocol**.



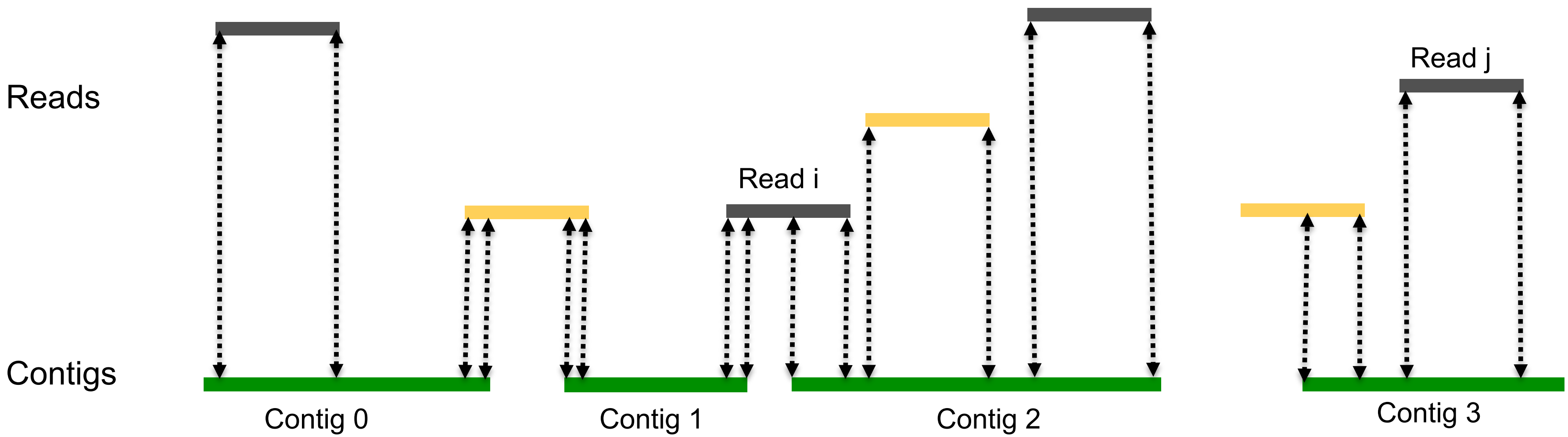
Lightweight synchronization protocol



Scaffolding: Main step maps reads to contigs

- **Input :** a set of reads and a set contigs
- **Output** : detailed alignments of the reads onto the contigs

Alignment with (large) input data; use of hash table to find possible matches



Read ID	Start-pos	End-pos	Contig ID	Start-pos	End-pos
Read i	1	4	Contig 1	152	155
Read i	130	150	Contig 2	1	21
Read j	1	150	Contig 3	101	250

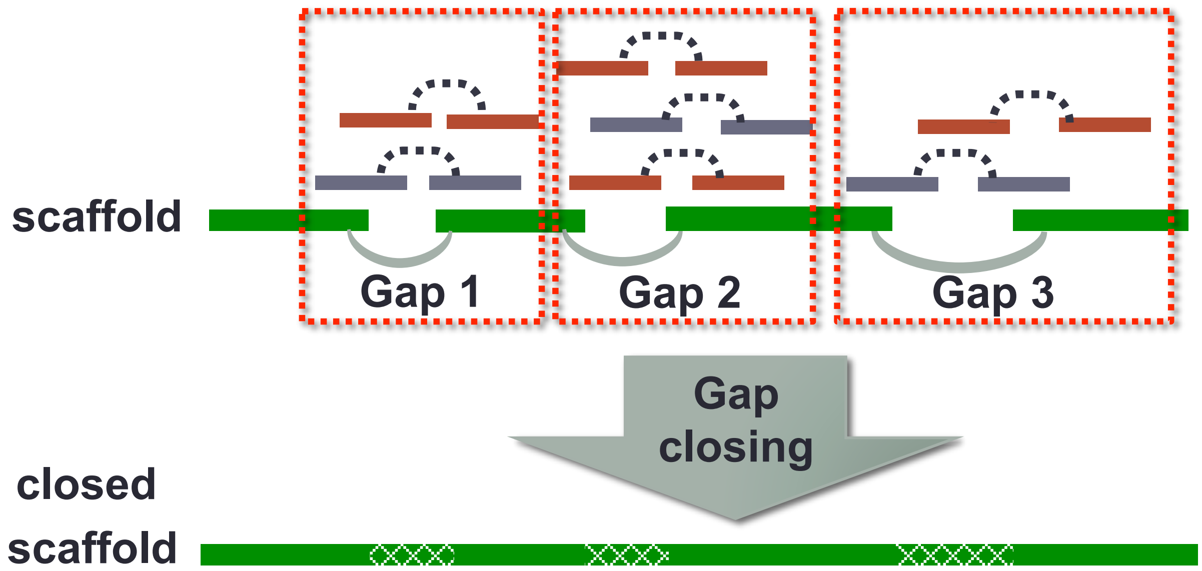
Seed-and-extend Algorithm Summary

1. Given a set of reference sequences (contigs), build an index of them using seeds (substrings) of fixed length **s**.

2. Given a query sequence (read), extract substrings of length **s** and look those up in the reference index → locate candidate contigs to be aligned.

3. Perform an extension algorithm (e.g. Smith-Waterman) on the read and the candidate contig to obtain detailed alignments.

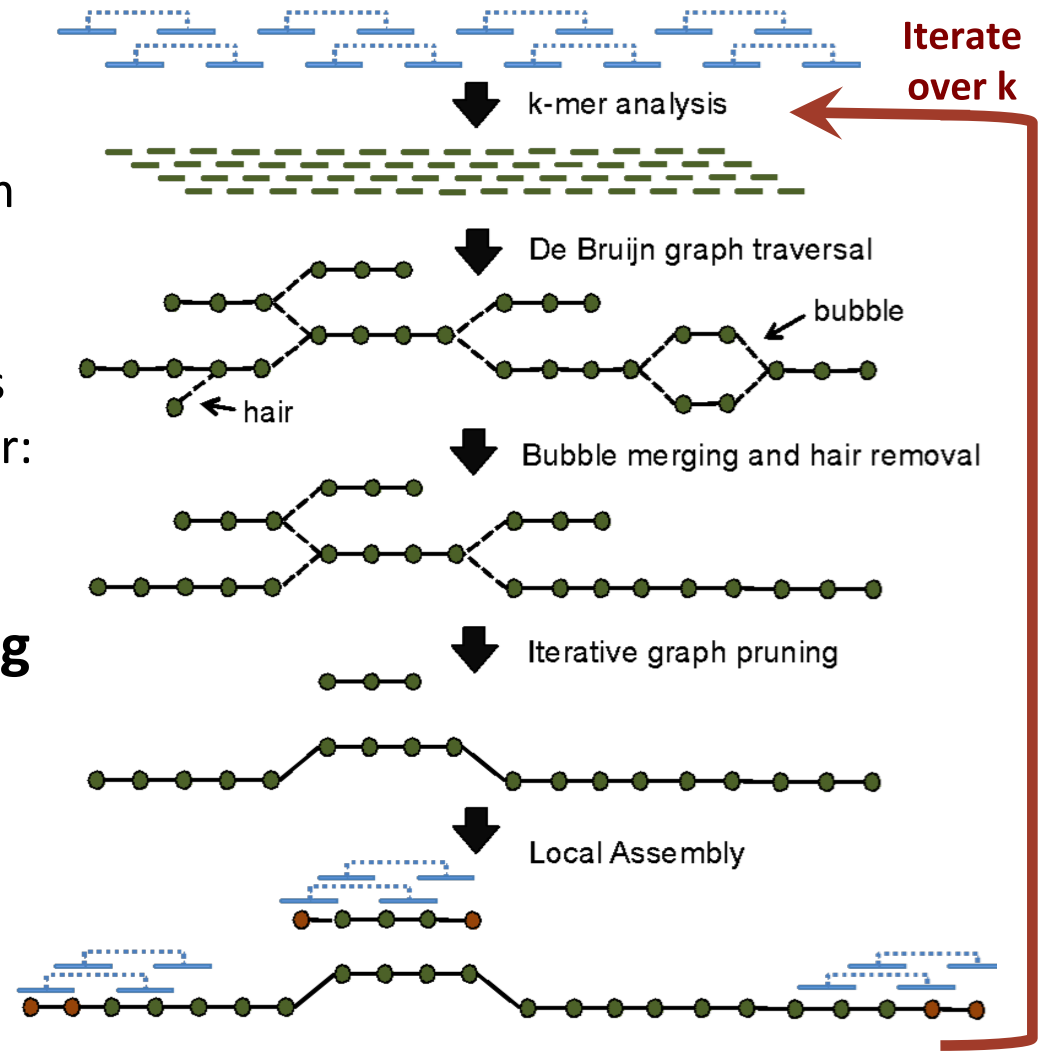
Gap Closing



- Gaps found from alignment and information about distance between paired reads
- Leads to load balancing problems (work stealing)

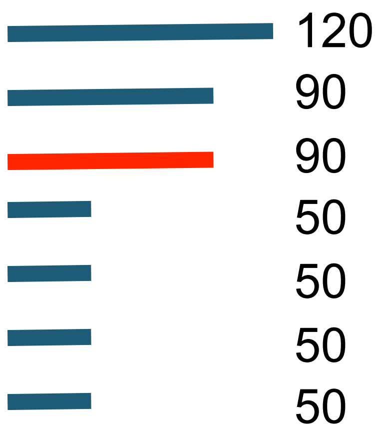
Extending HipMer for Metagenomes

- **Primary change in contig generation on k-mer graph**
  - Iterative contig generation, from small to large k-mer size
  - Small k: low coverage genomes
  - Large k: high-coverage genomes
  - Added/modified steps in HipMer: merging bubbles, iterative pruning, local assembly
- **Some changes to scaffolding**
  - Looping over scaffolding
  - Opportunities for future improvements
- **Scaffolding steps omitted from figure for simplicity**



Quality Metrics

- **A goal of this milestone was to identify good quality metrics**
- **Want to simultaneously maximize contig length and minimize errors**
  - –Sort output contigs by length and find the halfway point.
  - –Several standard metrics, simplest is N50.
  - –***N50*** is the minimal contig length X such that contigs of length at least x account for at least 50% of the total assembly length



Total assembly length = 500

First 3 contigs account for over half

N50 = 90 (length of 3rd contig)

Quality Metrics: Error-free contiguity

Contiguity

- ***N50*** is the contig length such that using longer or equal length contigs produces at least half (50%) the bases of the assembly.
- ***NG50*** is the contig length such that using longer or equal length contigs produces half (50%) the bases of the reference genome. This metric could be computed only if the reference is given.

Error-free contiguity

- ***NGA50*** similar to NG50, but uses lengths of aligned blocks rather than contigs. if a contig has a misassembly with respect to the reference, we break the contig into smaller pieces. Also, if a contig has bases that don't align to the reference, they are not counted in NGA50.
- **Median NGA50:** median NGA50 across all genomes for which NGA50 can be computed (requires sufficient coverage of genome)

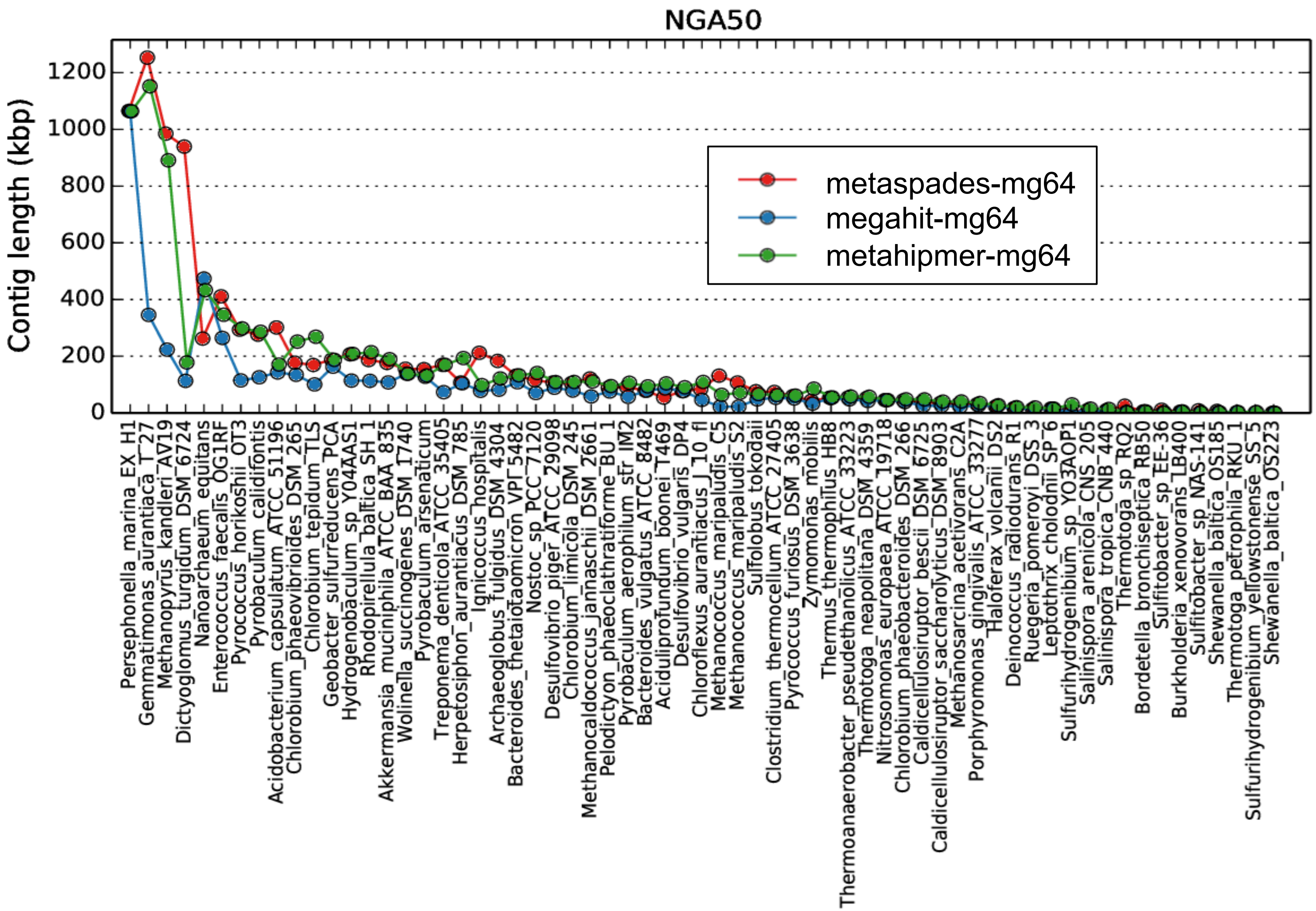
We will use NGA50 and Median NGA50 for error-free contiguity

Results for low complexity data (mg64)

Genome Statistics	MetaHipMer	metaSPAdes	MEGAHIT
Genome Fraction (%)	**95.5**	94.2	95.3
Misassemblies (↓)	456	**239**	309
Mismatches per 100 kpb (↓)	**45.6**	99.3	71.9
Median NGA50	**95129**	89288	58468
Predicted Genes	**202228**	191307	200367

- Low complexity dataset - only 64 genomes
- All assemblers find most genomes, with low errors
- MetaHipMer has more misassemblies due to small number of species (recall this is not normalized to assembly length)
- MetaHipMer has fewer mismatches, greater predicted genes and better NGA50

Error free contiguity (NGA50) on mg64



Results for medium complexity data (CAMI)

Genome Statistics	MetaHipMer	metaSPAdes	MEGAHIT
Genome Fraction (%)	66.3	68.7	**73.0**
Misassemblies (↓)	2031	**1210**	1579
Mismatches per 100 kpb (↓)	**80.3**	152.6	100.1
Median NGA50	**24906**	14219	16123
Predicted Genes	632459	635289	**637085**

- Medium complexity dataset - 225 genomes
- Lower coverage than mg64, with more errors
- MetaHipMer has a lower genome fraction, but better NGA50 than the others

Results for high complexity data (MC04)

Genome Statistics	**MetaHipMer**	**metaSPAdes**	**MEGAHIT**
Genome Fraction (%)	28.6	**32.7**	32.3
Misassemblies (↓)	12730	**9014**	17479
Mismatches per 100 kpb (↓)	**27.4**	70.3	67.9
Median NGA50	**15098**	10540	12744
Predicted Genes	**1386760**	1381479	1355607

- High complexity dataset - 800 genomes
- Low coverage from all assemblers due to challenging nature of dataset
- High rates of misassembly
- MetaHipMer has a lower genome fraction, but better NGA50 than the others

MetaHipMer Performance (preliminary results)

- **Performance on a mock community**
  - metaSPAdes: **11.5 hours** on a 32 core machine with 500 GB of RAM
  - metaHipMer: **17 minutes** on 80 Edison nodes (~2K cores) – **41x faster**

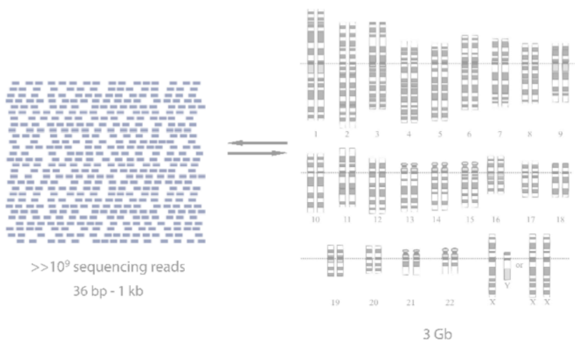
- metaSPAdes can't scale to the massive datasets (up to 80x larger)
  - For assembling TaraOceans dataset (8 TBytes, 80x larger), it would take **38 days**, on a machine with **40 TBytes** of memory.

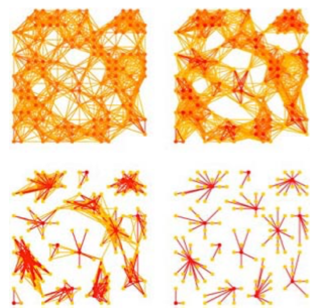
- metaHipMer modules scale to full machine !
  - Assuming 40% efficiency, the TaraOceans dataset could be assembled in < **1 hour** using full Edison

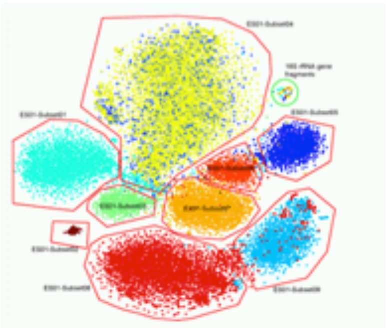
Summary on MetaGenomeAssembly

- **Assembly is an HPC problem**
  - (Meta)HipMer can handle previously unassembled data sets
- **Hardware support**
  - High injection rate RDMA communication; low latency; remote atomics
  - High bisection bandwidth (synchronous and asynchronous all-to-all)
- **Software support**
  - PGAS model for distributed data structures, one-sided communication
  - Hierarchical algorithms probably useful, but scaling well on/off node with UPC
- **Benchmarks and proxy applications**
  - NERSC-9 "Meraculous" benchmark is contig generation (graph/hash table)
  - Latency / bandwidth tests ("roofline like") for remote atomics, async all-to-all
- **MetaHipMer is a parameterized toolkit for HPC genome analysis**
  - Quality results comparable to other state-of-art assemblers, but can solve problems they cannot due to memory/performance scaling

ExaBiome: Exascale Solutions for Microbiome Analysis







Metagenome Assembly	Protein Clustering	Comparative Analysis
Graph algorithms, Hash Tables, alignment (Smith-Waterman)	Machine learning (clustering), sparse linear algebra / graphs	Alignment, Machine learning (dimensionality reduction), linear algebra
Fine-grained comm., all-to-all, remote atomics and fast I/O	Fast barriers, subset reductions	All-to-all

PI: Katherine Yelick (LBNL)

Discovering Biosynthetic Gene Clusters

"A **biosynthetic gene cluster** is a physically clustered group of two or more genes in a particular genome that *together encode a biosynthetic pathway* for the production of a specialized metabolite (including its chemical variants)"

[Medema et al. "Minimum information about a biosynthetic gene cluster." Nature chemical biology, 2015]

Input: Sparse matrix encoding genes and their nonzero pairwise similarities

Method: High-performance Markov clustering (HipMCL)

Desired scale: 10s of billions of genes, trillions of nonzero pairwise similarities

- Use proteins rather than genes (constant factor)
- Find connected components first (heuristic)
- Requires supercomputers

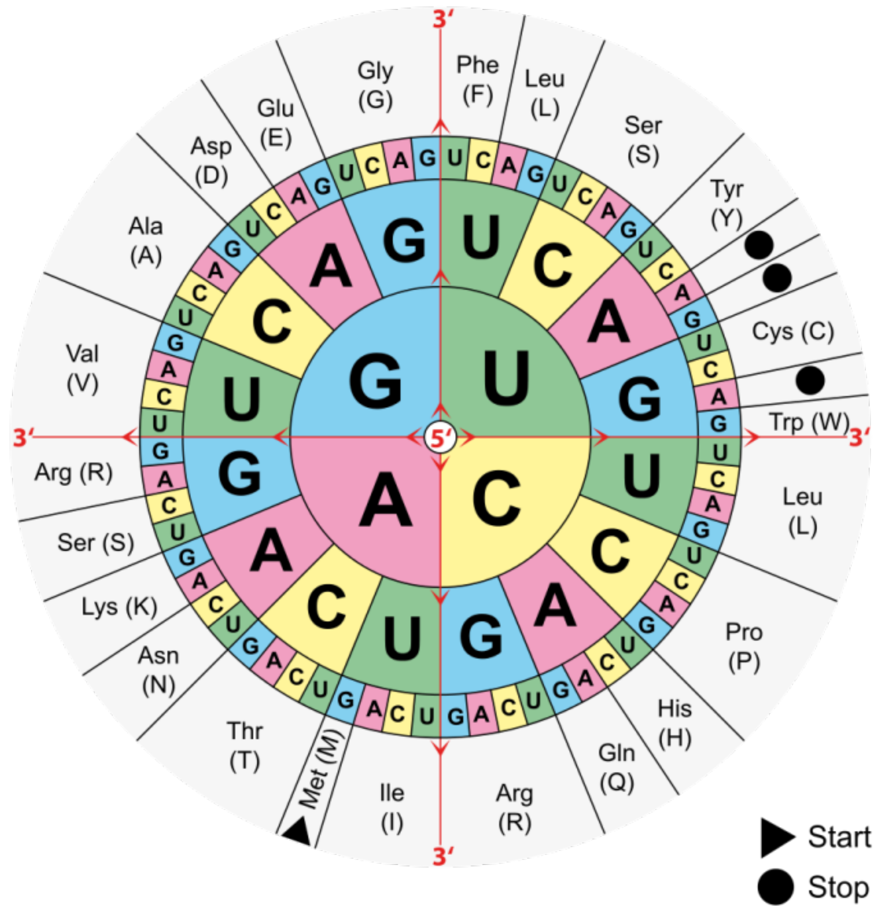
MCL is the de-facto algorithm in community for finding gene/protein families

"…MCL is remarkably robust to graph alterations…"

[Brohée, van Helden, "Evaluation of clustering algorithms for protein-protein interaction networks", 2006]

DNA to Proteins

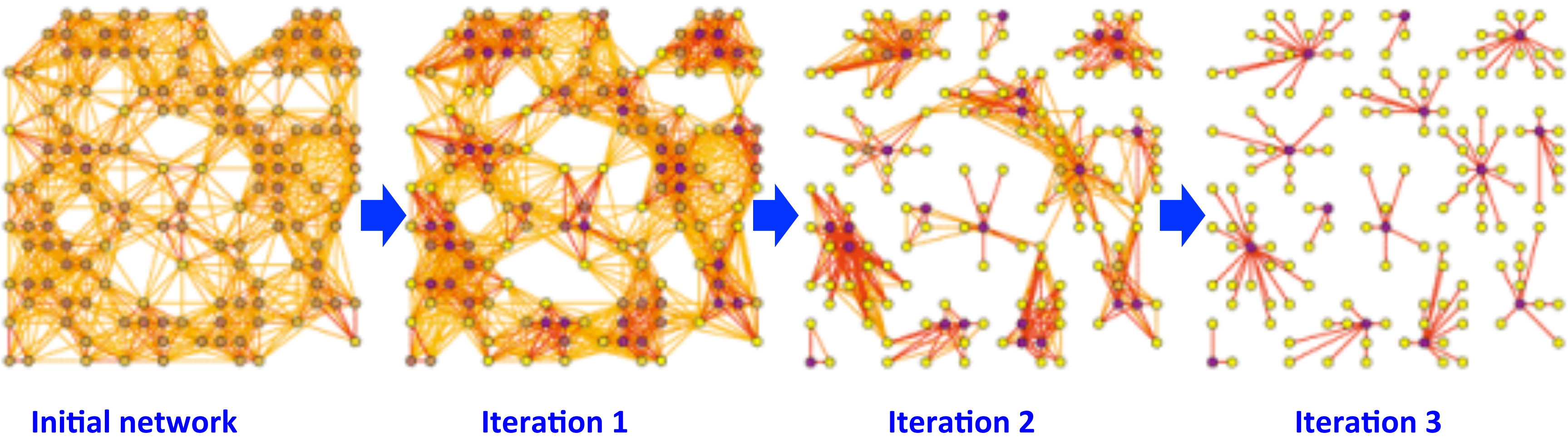
- DNA based on 4-character alphabet
- Three-letter codons represent one of the 20 regularly used amino acids
- Start and Stop codons mark the beginning/ end of genes



Markov Cluster algorithm

Input: Adjacency matrix A (sparse)

Image source: http://micans.org/mcl/



At each iteration:

Step 1 (Expansion): Squaring the matrix while
pruning (a) small entries, (b) denser columns

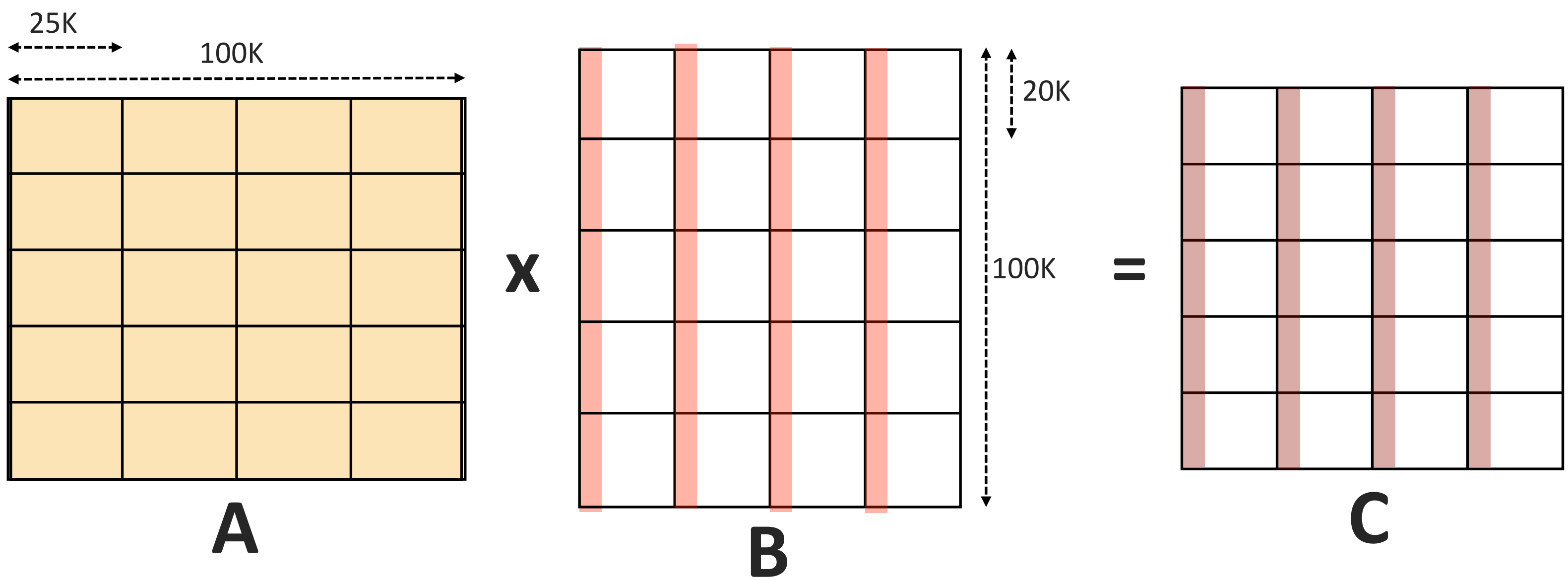
Naïve implementation: sparse matrix-matrix product (SpGEMM), followed by column-wise top-K selection and column-wise pruning

Step 2 (Inflation) : taking powers entry-wise

Scalable Distributed Memory, Memory Efficient SpGEMM

$\sqrt{p} \times \sqrt{p}$ Processor Grid

Split B into k pieces

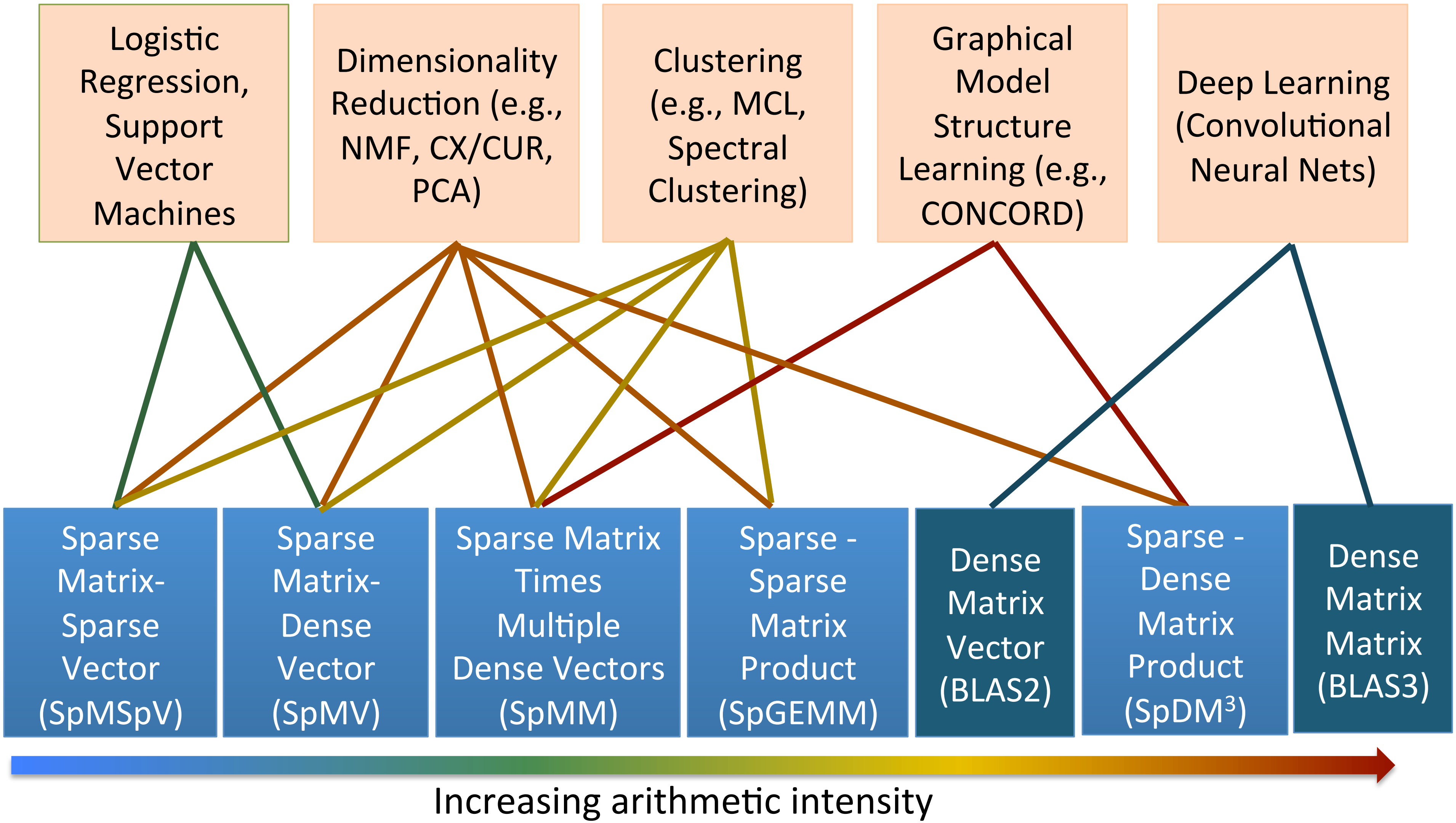


- Parts of the result is produced and pruned
- Memory requirement can be significantly reduced by increasing k
- However, A is needed to be broadcasted k times
- With k=20: MCL ran on 64 nodes of Cori in about 20 minutes

Challenges in distributed-memory MCL

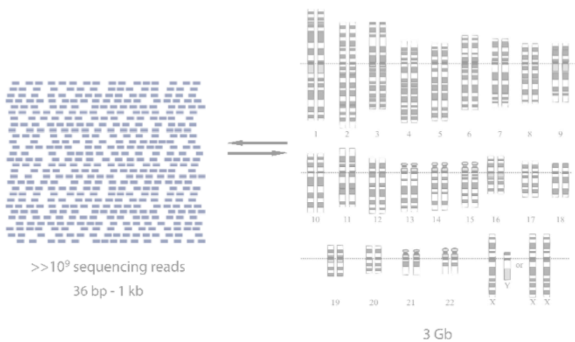
- **"Small" test dataset from Joint Genome Institute**
  – 47 Million proteins
  – 14 Billion interaction
  – 4.2 Trillion expected nonzeros in A^2
  – Memory requirement of naïve implementation: ~**100 TB**
- **Ultimate dataset is 1000 times larger.**
- **Memory efficient SpGEMM algorithm being developed**
  – Since the output is sparsified by column-wise pruning, we ***create A² part by part and prune on the fly*** to save memory.
  – Need to find ***the kth largest entry of each column at every iteration***: Trivial in single node, harder in distributed memory.
- **Interpretation of final clusters need distributed-memory connected components**
  – Not a bottleneck: cheaper and only done once (not per iteration)

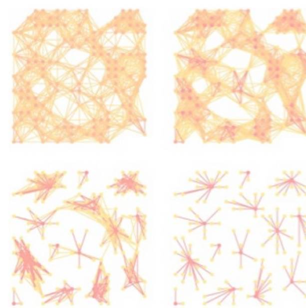
Machine Learning Mapping to Linear Algebra in General

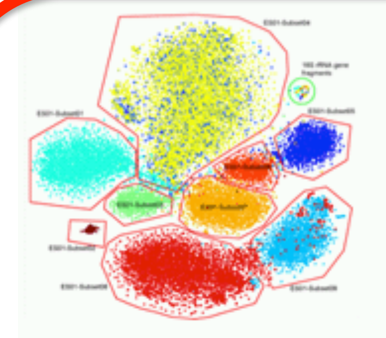


Aydin Buluc, Sang Oh, John Gilbert, Kathy Yelick

ExaBiome: Exascale Solutions for Microbiome Analysis







Metagenome Assembly	Protein Clustering	Comparative Analysis
Graph algorithms, Hash Tables, alignment (Smith-Waterman)	Machine learning (clustering), sparse linear algebra / graphs	Alignment, Machine learning (dimensionality reduction), linear algebra
Fine-grained comm., all-to-all, remote atomics and fast I/O	Fast barriers, subset reductions	All-to-all

PI: Katherine Yelick (LBNL)

Metagenome composition description

Three approaches to describe and analyze the metagenome sequencing dataset:

"features"

1) taxonomic composition-based (who is in the metagenome)
2) functional annotation-based (what can this metagenome do)
3) k-mers (shingles) composition-based (i.e., shingle frequency vector-based)

K-mer based approach (MASH) is:

– More resilient to errors /degradation of DNA, coverage bias, composition, short and redundant fragments

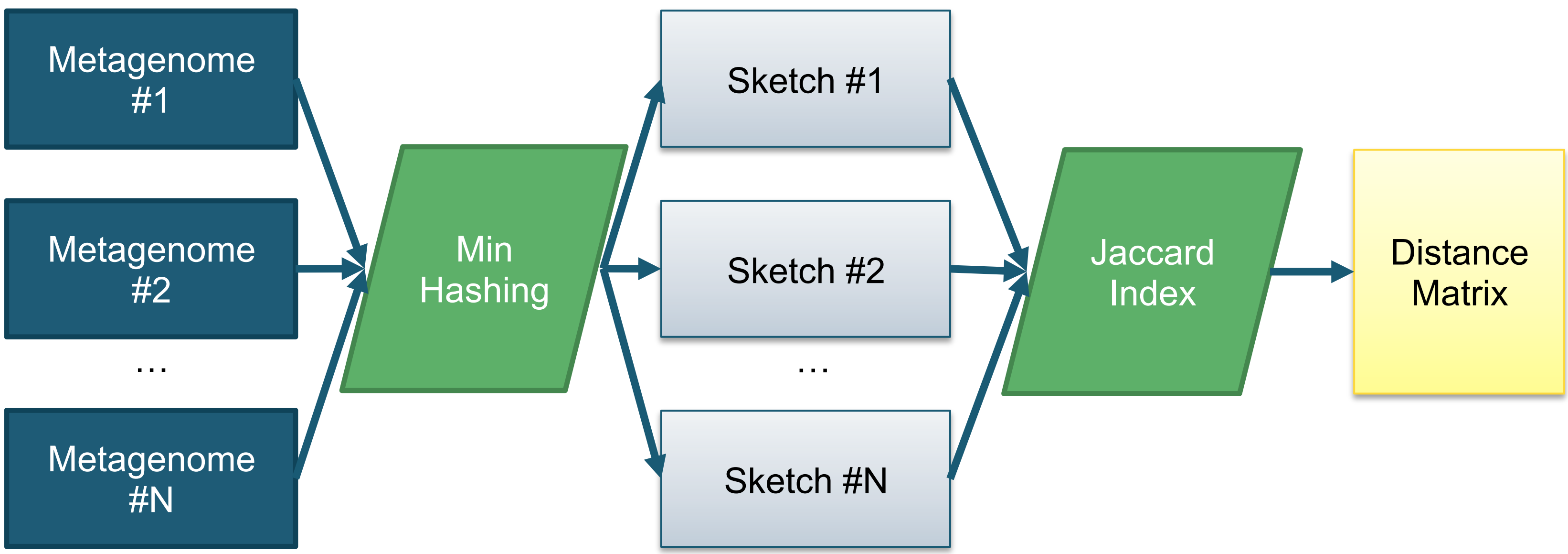
Taxonomic (GOTTCHA) or functional approaches:

– Fast and memory efficient; also gives some semantic information about differences

MASH – an alternate MinHash based approach

- **"…extends MinHash dimensionality-reduction technique to include a pairwise mutation distance and *P* value significance test, enabling the efficient clustering and search of massive sequence collections…"**

Ondov et al. Genome Biology (2016) 17:132 DOI 10.1186/s13059-016-0997-x



MASH distance is sufficiently accurate

Ondov et al. Genome Biology (2016) 17:132 DOI 10.1186/s13059-016-0997-x

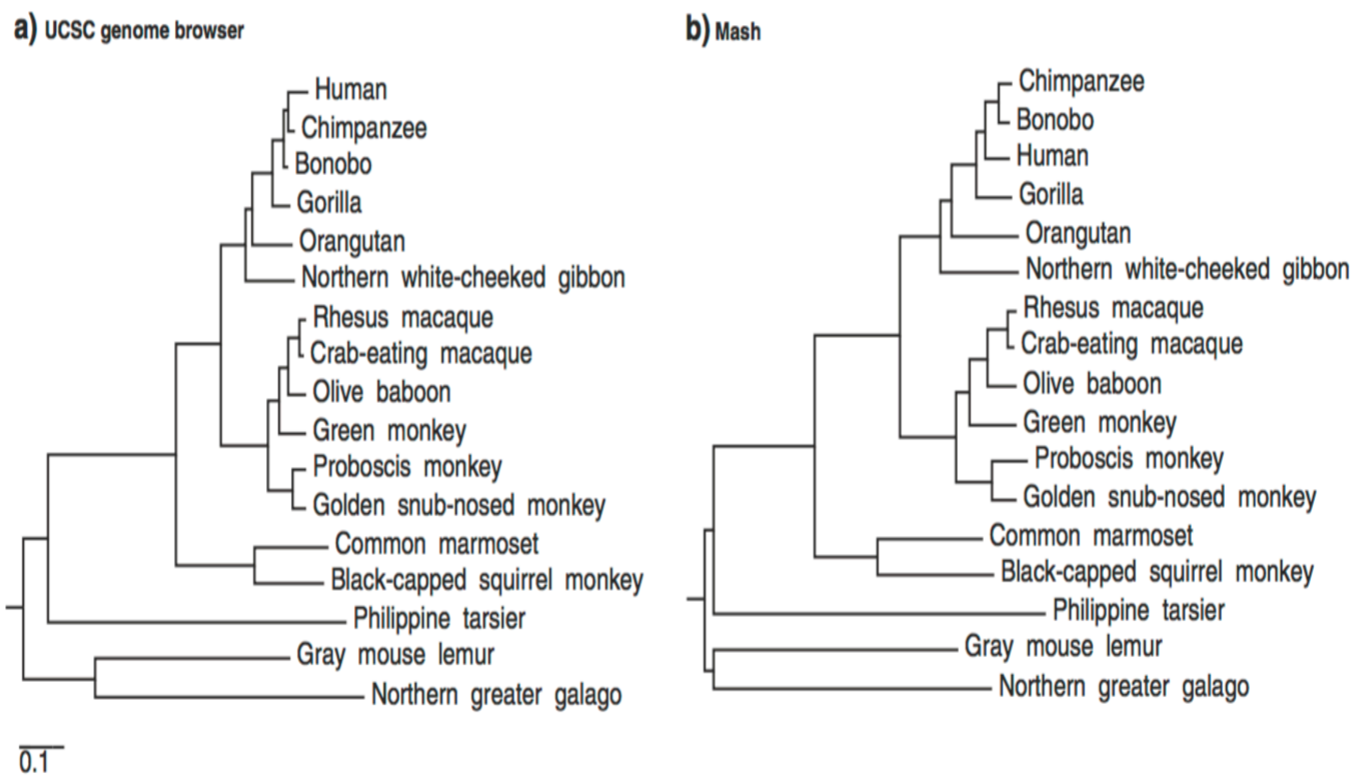
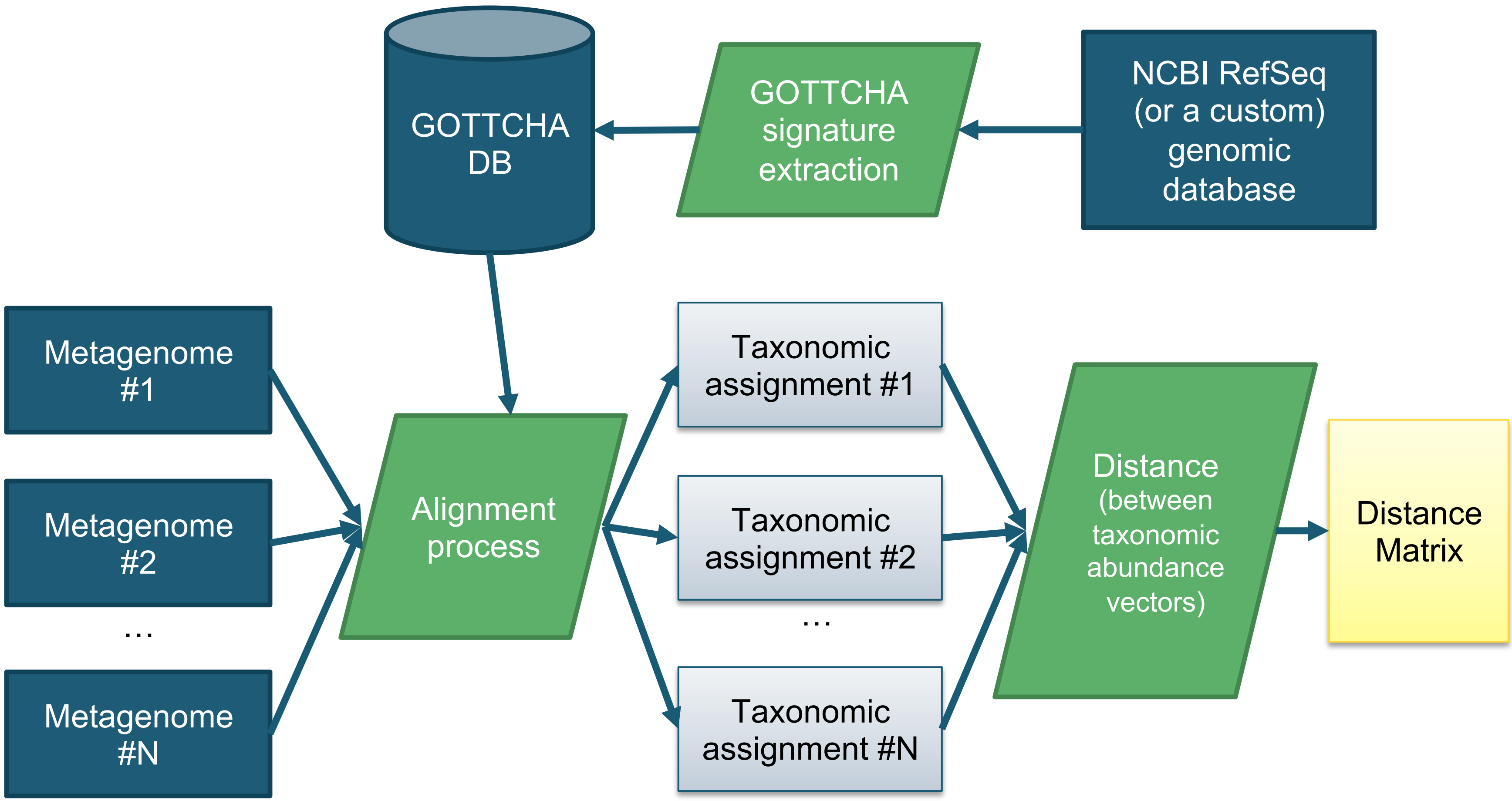


Fig. 4 Primate trees from the UCSC genome browser and Mash. **a** A primate phylogenetic tree model from the UCSC genome browser, with branch lengths derived from fourfold degenerate sites extracted from reference gene multiple alignments. **b** A comparable Mash-based tree generated from whole genomes using a sketch size of $s = 1000$ and k-mer size of $k = 21$. Additional file 1: Figure S6 includes this Mash tree with five additional mammals of increasing divergence

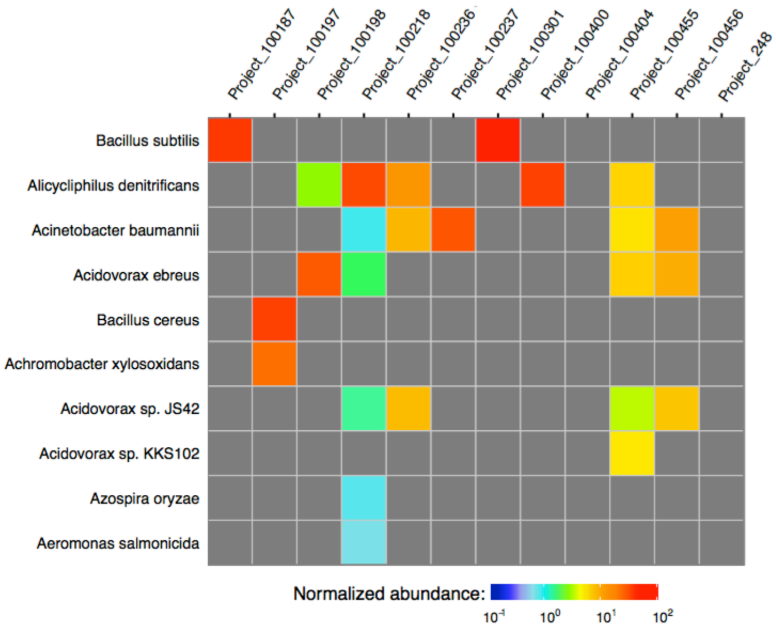
MASH pros and cons

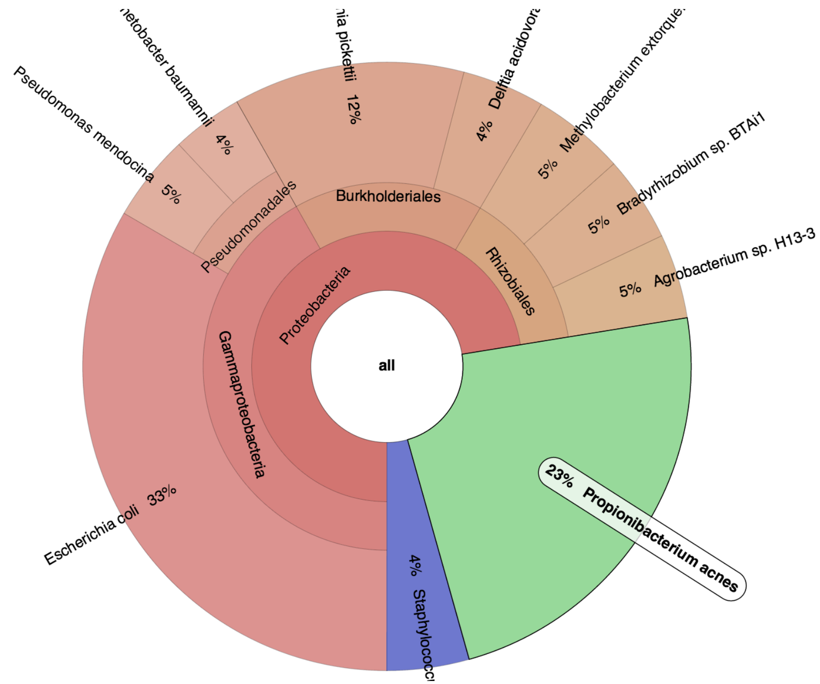
- **+ REALLY fast (just four simple consecutive steps).**
  - –K-mer extraction -> MinHash -> sketch -> distance
- **+ Reduces metagenome data size significantly**
  - –|sketch|<<|metagenome|
- **+Can cluster metagenomes with a lot of unknowns (un-annotatable)**
- **–There is a threshold in k-mer and sketch sizes which affects the tradeoff between specificity/sensitivity and the computational complexity/space requirements**
  - –The threshold can be estimated if the divergence between metagenomes can be roughly approximated
- **–Provides *no insight* into the genome taxonomic or functional composition**

GOTTCHA - Genome Signature Discovery



GOTTCHA – an interactive comparative visual display





GOTTCHA2 implementation

- Developed at LANL, will be released with GPL-2
- The database creation tool is not yet publicly available, coded in MPI C/C++ (we distribute binary database files)
- The taxonomic assignment pipeline is GPL-2 open sourced https://github.com/poeli/GOTTCHA2
- Currently relies on BWA for alignment https://github.com/lh3/bwa, distributed under GPL-3, we are working on the MerAligner integration from MetaHipMer

Summary on Post-Assembly Analytics

Finding gene clusters related groups (ML – clustering)

- Large graph / sparse matrix problems (sparse matrix-matrix multiply)
- Needs memory-efficient as well as scalable distributed memory approach

- **Hardware support (preliminary)**
  - High bisection bandwidth; low latency; fast I/O
- **Software support**
  - Sparse matrix and graph libraries (e.g., Graph BLAS approach)
  - MPI (bulk synchronous so far)

Comparative metagenome analysis (ML – dimensionality reduction)

- Dominated by alignment; see MerAligner discussion
- Also statistical (counting) and sparse matrix algorithms

ExaBiome: Exascale Solutions for Microbiome Analysis (1.2.1.20)

Application Domain

- **Application Area:** Microbiomes are integral to the environment, health, and biomanufacturing. They occur in communities that are collected and sequenced as a group, called metagenomes. These metagenomes lead to some of the most computationally demanding problems in bioinformatics, including assembly, protein clustering and comparative whole-metagenome analysis.
- **Challenge Problem**:
- *4 Year:* Provide scalable tools for and complete assembly and analysis of metagenome data in SRA and IMG.
- *10 Year:* Perform high quality assembly and analysis of 1 million metagenomes on an early exascale system to reveal microbial functions for environmental remediation and other applications.

Physical Models and Code(s)

- **Physical Models**: *Assembly of genomes using DNA sequence fragments from metagenomes. Clustering of genes culled from metagenomes to identify novel protein families. Comparative analysis across whole metagenomes.*
- **Codes**: (Meta)*HipMer, HipMCL, Mash, GOTCCHA, Combinatorial BLAS*
- **Motifs:** *Graph Traversal, Sparse Matrices, Dynamic Programming, (possibly Graphical Models)*

Partnerships

Co-Design Centers:

- *ExaGraph (if funded) and possibly AMReX*

Software Technology Centers:

- *Programming: PAGODA, PROTEAS, Legion, ExaMPI and OMPI-X*
- *Performance/Systems: HPCToolkit, PAPI, Qthreads, ExaHDF5*
- *Libraries: xSDK4ECP, Sparse Solvers, Trilinos*

Application Projects:

- *Cancer, NWChem*

First Year Development Plans

Develop high quality, scalable metagenome assembly

- *Extend HipMer with features for metagenomes, now called MetaHipMer (currently a research prototype)*
- *Evaluate and improve quality and scalability of MetaHipMer; compare to other metagenome assemblers*
- *Release MetaHipMer*

Use machine learning algorithms for metagenome analysis

- *Develop and demonstrate HipMCL clustering algorithm for genes from metagenomes*
- *Evaluate use of MerAligner (a phase in MetaHipMer) for GOTTCHA's metagenome characterization*

PI