www.bsc.es


Barcelona Supercomputing Center
*Centro Nacional de Supercomputación*

# Porting Genomics Workflows to the Parallel In-Memory Database (PIMD)

Autonomic Systems and eBusiness Platforms

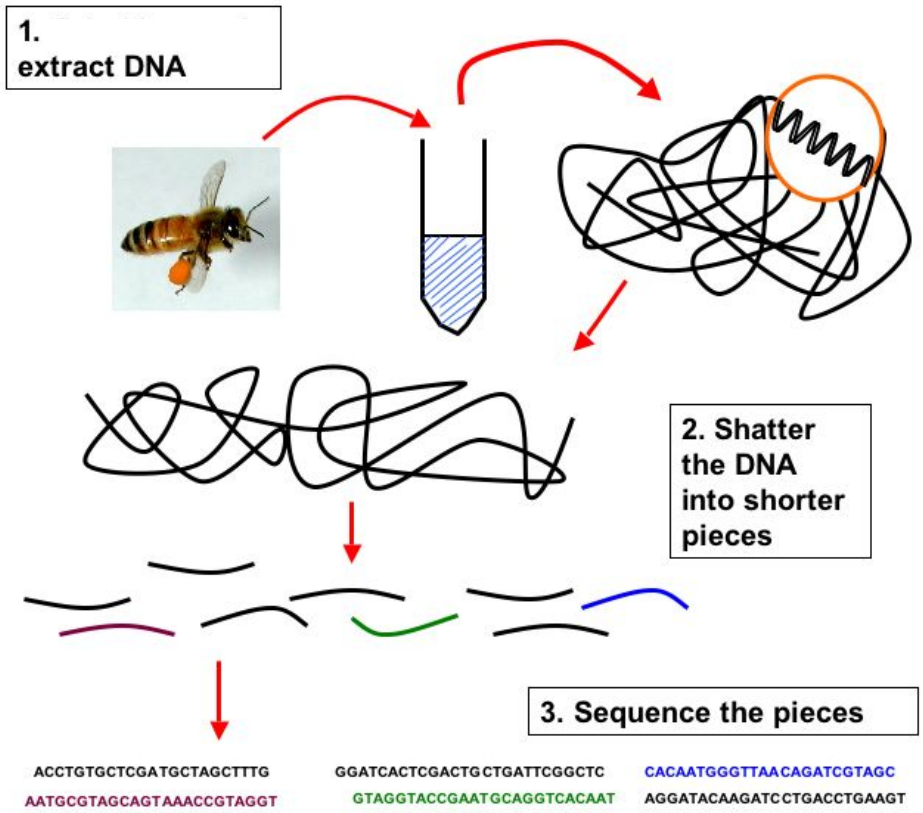
David Carrera
david.carrera@bsc.es

# Index

- Genomics Background
- The Workflow
- Data Structures in BWA Aligner
- PIMD: Parallel In-Memory Database
- Porting the Workflow


BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación
GENOMICS BACKGROUND

# What is Genome Alignment?



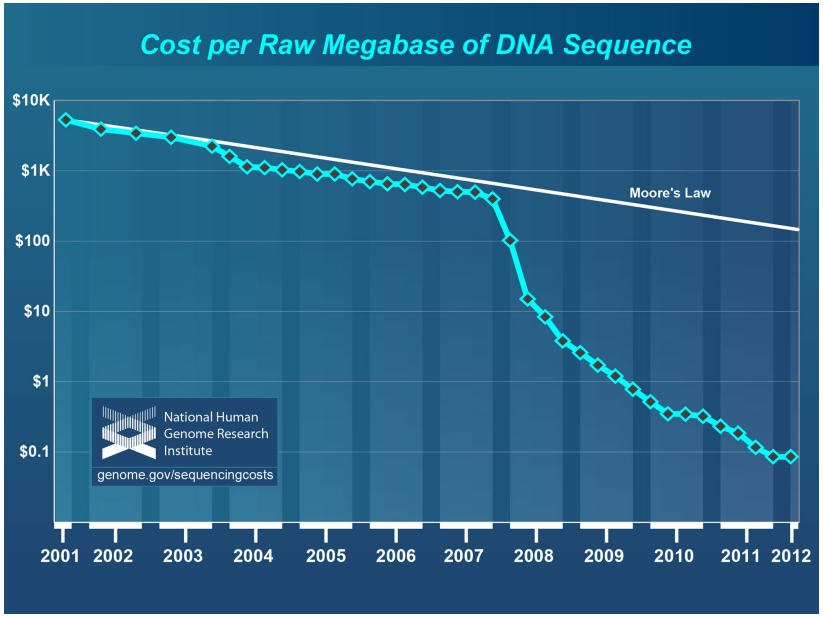

Sequence Assembly

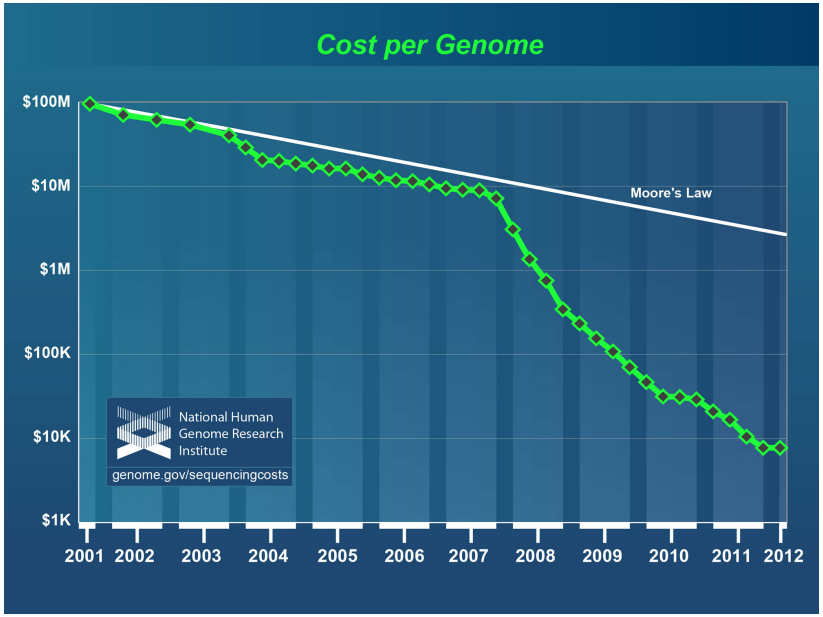

Read Alignment

# Current computer systems in use for bioinformatics jobs

- Current computer systems available at genomics research institutions are commonly designed to run CPU-bound.
  - Having 24 or 32 GB of RAM per core is not unusual in some cases, as sequence alignment applications have large requirements of memory
- Large shared storage for all nodes and users.
  - Use of a distributed file system like GPFS
- GbitE network communicates all nodes of the system.
- FASTA, FASTQ, SAM, BAM… Mostly record-oriented, Many ASCII-based

# Sequencing Costs



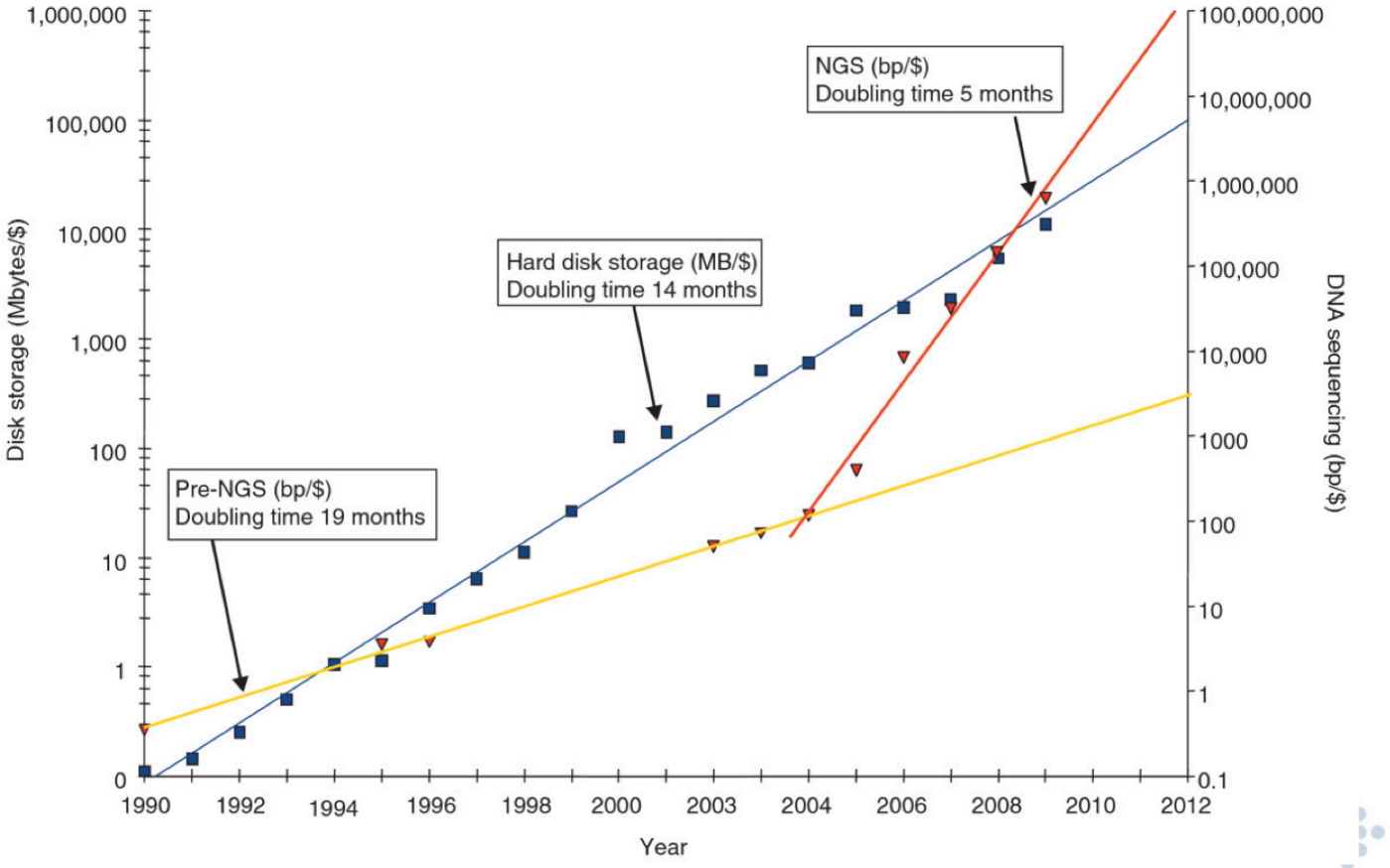

Source: **National Human Genome Research Institute** (NHGRI)
http://www.genome.gov/sequencingcosts/

(1) "Cost per Megabase of DNA Sequence" — the cost of determining one megabase (Mb; a million bases) of DNA sequence of a specified quality
(2) "Cost per Genome" - the cost of sequencing a human-sized genome. For each, a graph is provided showing the data since 2001

In both graphs, the data from 2001 through October 2007 represent the costs of generating DNA sequence using Sanger-based chemistries and capillary-based instruments ('first generation' sequencing platforms). Beginning in January 2008, the data represent the costs of generating DNA sequence using 'second-generation' (or 'next-generation') sequencing platforms. The change in instruments represents the rapid evolution of DNA sequencing technologies that has occurred in recent years.

# Sequencing Costs

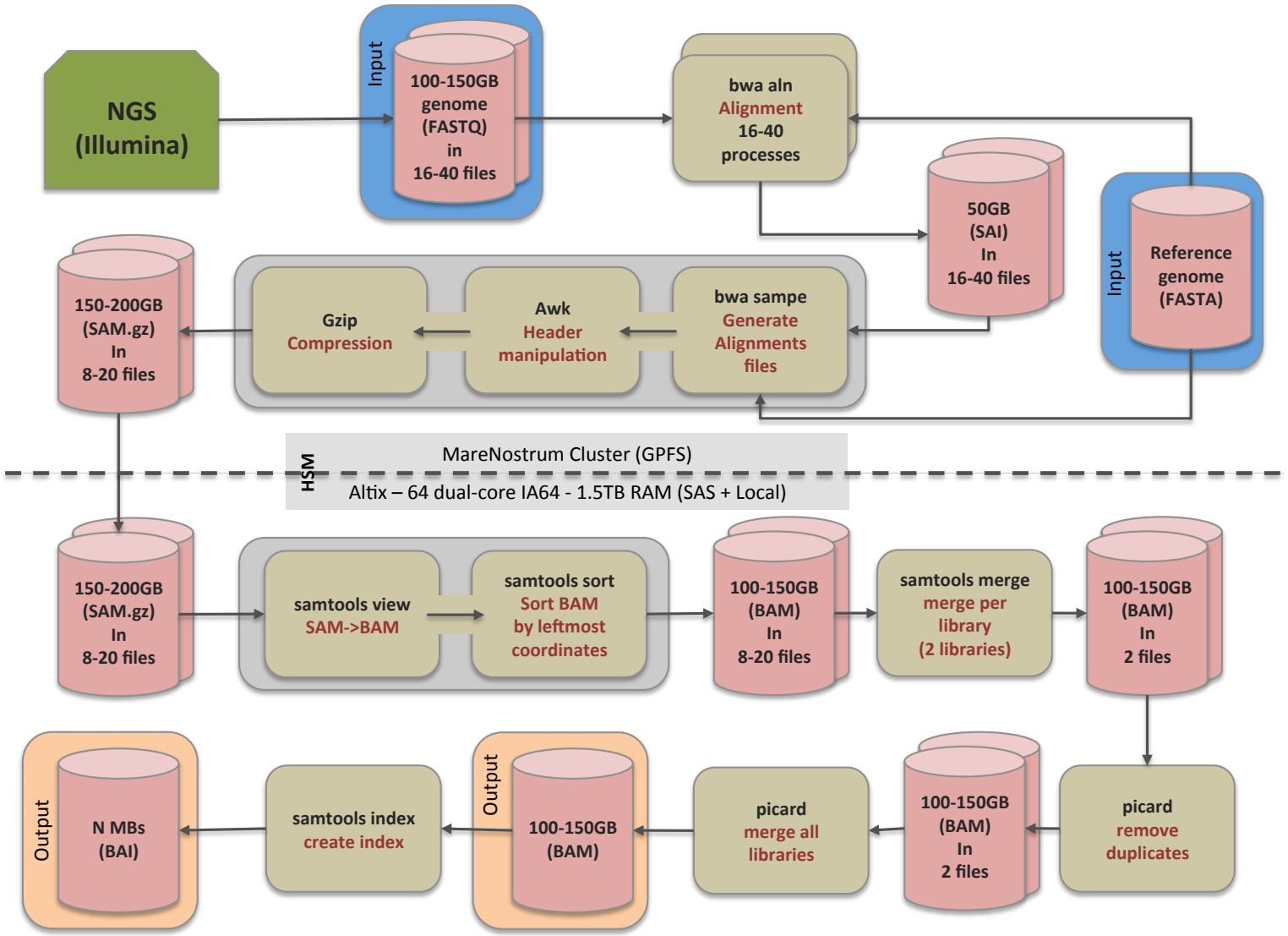
## NextGen Sequencing a Game-Changer




We have an issue…. Need for faster ways of processing data, not storing everything…


BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación
THE WORKFLOW

# Genomics Workflow

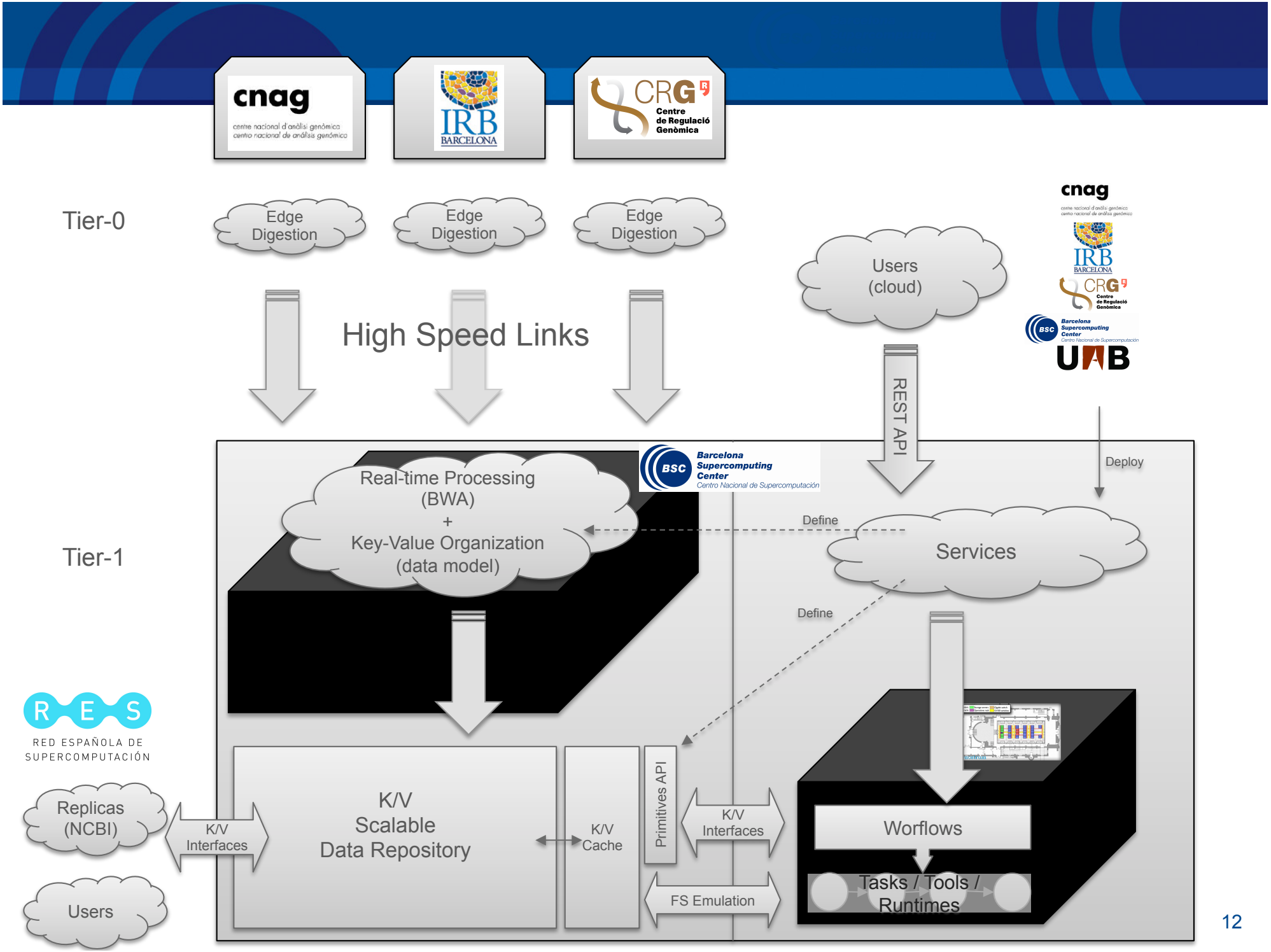


# Potential benefits (short term)

- Minimize time loading/storing flat files
  - Current tools use multithreading but each process/node keeps all data in memory and dumps to a local file that requires merging
- Data pipelining easier
  - One experiment runs while the next is uploading data (reads/reference)
- Simple order (on insertion) and merging
- Keep data structured
  - Merging in many cases can be avoided selecting the right “key”
  - Experiment data management DB-like



11


cnag
IRB BARCELONA
CRG Centre de Regulació Genòmica
Tier-0
Edge Digestion
Edge Digestion
Edge Digestion
Users (cloud)
cnag
IRB BARCELONA
CRG Centre de Regulació Genòmica
Barcelona Supercomputing Center
UAB
High Speed Links
REST API
Deploy
Tier-1
Real-time Processing (BWA) + Key-Value Organization (data model)
BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación
Define
Services
Define
R E S
RED ESPAÑOLA DE SUPERCOMPUTACIÓN
Replicas (NCBI)
Users
K/V Interfaces
K/V Scalable Data Repository
K/V Cache
Primitives API
K/V Interfaces
FS Emulation
Worflows
Tasks / Tools / Runtimes


# DATA STRUCTURES IN BWA ALIGNER

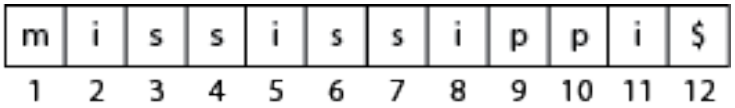
# The road to the Ferragina Manzini Index

- Problem:
  - 4Gb in a reference genome (4 billion chars in a sequence, Σ=4)
  - 20 -100 bases in a short read
  - Billions of reads
  - Align each one in the 4Gb reference genome sequence

- An FM-index is a compressed full-text substring index based on the Burrows-Wheeler transform, with some similarities to the suffix array. It was created by Paolo Ferragina and Giovanni Manzini, who describe it as an opportunistic data structure as it allows compression of the input text while still permitting fast substring queries

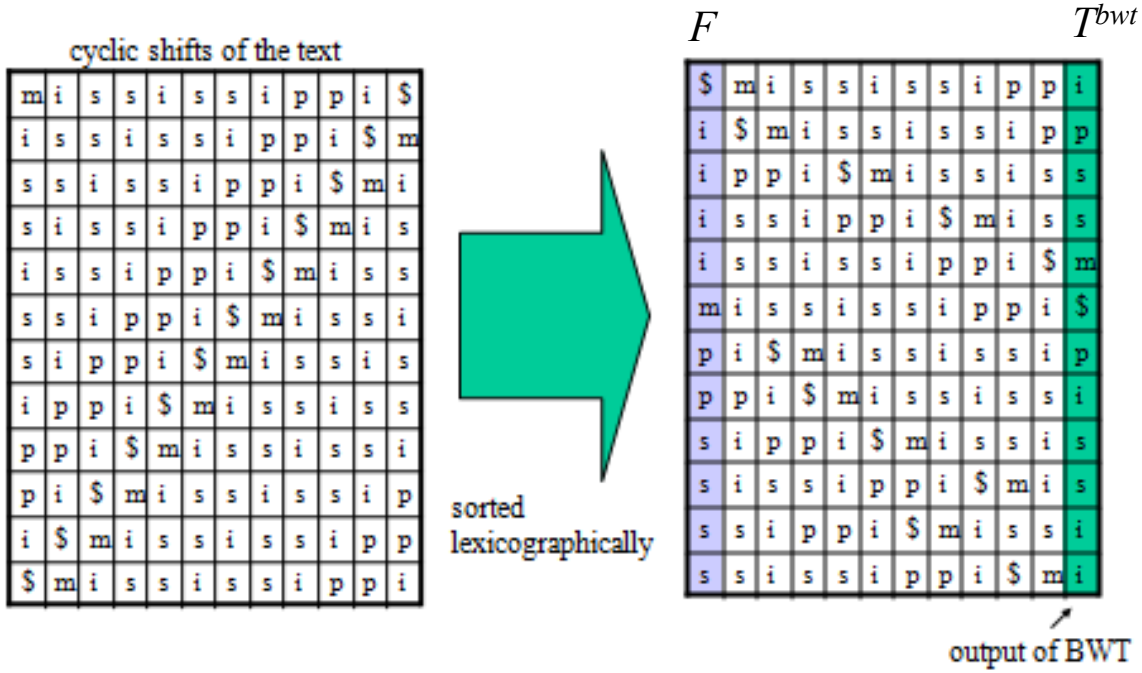
- **Search cost independent of the reference length or Σ, and linear with the length of the query**
  - *O(n) for a substring of len n*

# Background: SA




| SA | | | SA | |
|---|---|---|---|---|
| 1 | mississippi$ | | 12 | $ |
| 2 | ississippi$ | | 11 | i$ |
| 3 | ssissippi$ | | 8 | ippi$ |
| 4 | sissippi$ | | 5 | issippi$ |
| 5 | issippi$ | | 2 | ississippi$ |
| 6 | ssippi$ | sort → | 1 | mississippi$ |
| 7 | sippi$ | | 10 | pi$ |
| 8 | ippi$ | | 9 | ppi$ |
| 9 | ppi$ | | 7 | sippi$ |
| 10 | pi$ | | 4 | sissippi$ |
| 11 | i$ | | 6 | ssippi$ |
| 12 | $ | | 3 | ssissippi$ |

# Background: BWT




| BWT | SA | |
|---|---|---|
| i | 12 | $ |
| p | 11 | i$ |
| s | 8 | ippi$ |
| s | 5 | issippi$ |
| m | 2 | ississippi$ |
| $ | 1 | mississippi$ |
| p | 10 | pi$ |
| i | 9 | ppi$ |
| s | 7 | sippi$ |
| s | 4 | sissippi$ |
| i | 6 | ssippi$ |
| i | 3 | ssissippi$ |

Burrows-Wheeler Transform can be also **derived from SA**

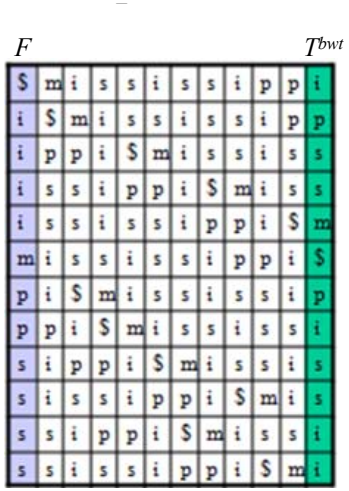
SA[i] != 1: BWT[i] = S[SA[i] – 1]
else : BWT[i] = $

# Background: FM Index

$$s' = C[P[i]] + rank(s-1, P[i]) + 1$$
$$e' = C[P[i]] + rank(e, P[i])$$

Backward search uses:

BWT <- Derived from SA
C[c]: Total count of occurrences of ‘c’ in S
Rank(i,c): Occurrences of ‘c’ in BWT[0,i]



$T^{bwt}$

$C$ table

| $ | $i$ | $m$ | $p$ | $s$ |
|---|---|---|---|---|
| 0 | 1 | 5 | 6 | 8 |

Occ($c$,$q$)

| $ | $i$ | $m$ | $p$ | $s$ |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 2 |
| 0 | 1 | 1 | 1 | 2 |
| 1 | 1 | 1 | 1 | 2 |
| 1 | 1 | 1 | 2 | 2 |
| 1 | 2 | 1 | 2 | 2 |
| 1 | 2 | 1 | 2 | 3 |
| 1 | 2 | 1 | 2 | 4 |
| 1 | 3 | 1 | 2 | 4 |
| 1 | 4 | 1 | 2 | 4 |

# Background: FM Index example

Objective: find ‘iss’ in S

| | | F | BWT | SA | | |
|---|---|---|---|---|---|---|
| 1 | ▶ | $ | i | 12 | $ | s = 1 |
| 2 | | i | p | 11 | i$ | e = 12 |
| 3 | | i | s | 8 | ippi$ | |
| 4 | | i | s | 5 | issippi$ | |
| 5 | | i | m | 2 | ississippi$ | |
| 6 | | m | $ | 1 | mississippi$ | |
| 7 | | p | p | 10 | pi$ | |
| 8 | | p | i | 9 | ppi$ | |
| 9 | | s | s | 7 | sippi$ | |
| 10 | | s | s | 4 | sissippi$ | |
| 11 | | s | i | 6 | ssippi$ | |
| 12 | ▶ | s | i | 3 | ssissippi$ | |

$$s' = C[c] + rank(0, c) + 1 = 8 + 0 + 1 = 9$$
$$e' = C[c] + rank(12, c) = 8 + 4 = 12$$

| | | F | BWT | SA | | |
|---|---|---|---|---|---|---|
| 1 | | $ | i | 12 | $ | s = 9 |
| 2 | | i | p | 11 | i$ | e = 12 |
| 3 | | i | s | 8 | ippi$ | |
| 4 | | i | s | 5 | issippi$ | |
| 5 | | i | m | 2 | ississippi$ | |
| 6 | | m | $ | 1 | mississippi$ | |
| 7 | | p | p | 10 | pi$ | |
| 8 | | p | i | 9 | ppi$ | |
| 9 | ▶ | s | s | 7 | sippi$ | |
| 10 | | s | s | 4 | sissippi$ | |
| 11 | | s | i | 6 | ssippi$ | |
| 12 | ▶ | s | i | 3 | ssissippi$ | |

$$s'' = C[c] + rank(8, c) + 1 = 8 + 2 + 1 = 11$$
$$e'' = C[c] + rank(12, c) = 8 + 4 = 12$$

# Background: FM Index example

| | | F | BWT | SA | | |
|---|---|---|---|---|---|---|
| 1 | | $ | i | 12 | $ | s = 11 |
| 2 | | i | p | 11 | i$ | e = 12 |
| 3 | | i | s | 8 | ippi$ | |
| 4 | | i | s | 5 | issippi$ | |
| 5 | | i | m | 2 | ississippi$ | |
| 6 | | m | $ | 1 | mississippi$ | |
| 7 | | p | p | 10 | pi$ | |
| 8 | | p | i | 9 | ppi$ | |
| 9 | | s | s | 7 | sippi$ | |
| 10 | | s | s | 4 | sissippi$ | |
| 11 | ▶ | s | i | 6 | **ss**ippi$ | |
| 12 | ▶ | s | i | 3 | **ss**issippi$ | |

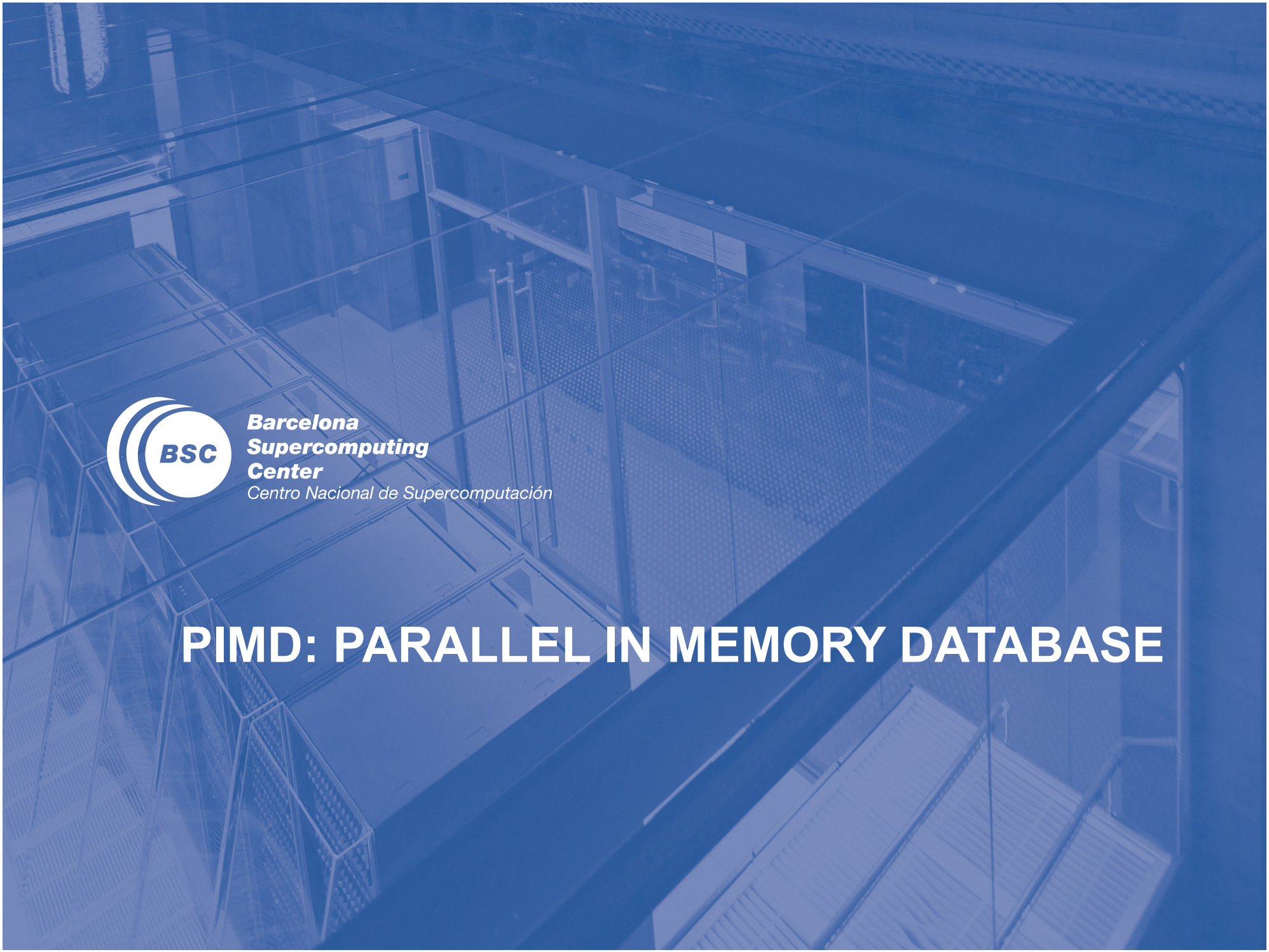
$$s''' = C[c] + rank(10, c) + 1 = 1 + 2 + 1 = 4$$
$$e''' = C[c] + rank(12, c) = 1 + 4 = 5$$

| | | F | BWT | SA | | |
|---|---|---|---|---|---|---|
| 1 | | $ | i | 12 | $ | s = 4 |
| 2 | | i | p | 11 | i$ | e = 5 |
| 3 | | i | s | 8 | ippi$ | |
| 4 | ▶ | i | s | 5 | **iss**ippi$ | |
| 5 | ▶ | i | m | 2 | **iss**issippi$ | |
| 6 | | m | $ | 1 | mississippi$ | |
| 7 | | p | p | 10 | pi$ | |
| 8 | | p | i | 9 | ppi$ | |
| 9 | | s | s | 7 | sippi$ | |
| 10 | | s | s | 4 | sissippi$ | |
| 11 | | s | i | 6 | ssippi$ | |
| 12 | | s | i | 3 | ssissippi$ | |

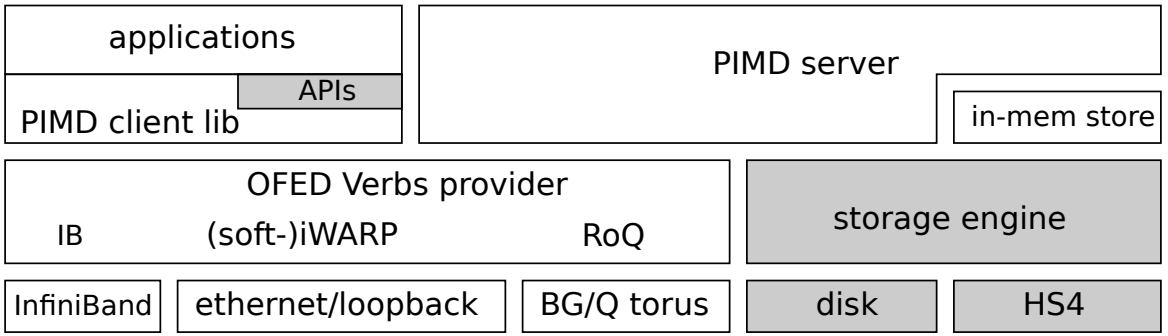
# Inexact matching

- Note that results in opening a tree of options at each step
- Consider:
  - Insertions
  - Deletions
  - Mutations
  - …
- Pruning based on heuristics
- Different heuristics for each tool
- Results only comparable for the same tool and parameters
  - No semantics modifications is a must for porting

- Need for new designed tools is a different problem


BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación
PIMD: PARALLEL IN MEMORY DATABASE

# PIMD: Stack and Operations

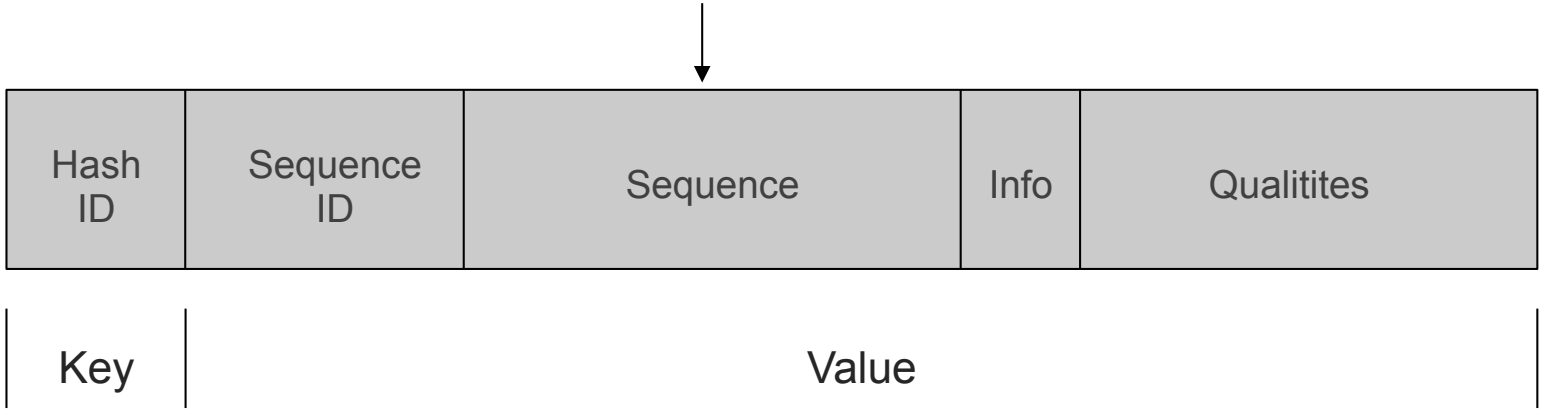



- Operations:
  - Init
  - Connect / Disconnect
  - Open / Close a Partitioned Data Set (PDS) – NameSpace
  - Insert / iInsert (including PSD, key, keyLen, value, valueLen, offset)
  - Retrieve / iRetrieve (including PSD, key, keyLen, value, valueLen, offset)
  - Wait, WaitAny, WaitAll
  - LocalCursor (keys only in one node): Open, GetNextElement, Close
  - GlobalCursor (keys in all the namespace): Open, GetNextElement, Close
  - BulkInserts

# Storing alignments (DB-like)

- TupleStash provides a tuple wrapper with high performance accesor methods
  - Also provides tuple operators that can be used to manage data ‘close to the data’
- Simple case: FASTQ data (mostly short lived)

```
@ERR023143.1 IL10_3212:5:1:12:1188/1
CAATGGAAGTCAANGGTTACAGGCTTGCAGATTACTTTAAAATATCATCNNNNNNNNNNNNNNNNNNNNNNNNNNN
+
BBBCBCBBCBBB4%-AABBBBCCCBB,*4>::?A4A@B>BABBBA>@@2%%%%%%%%%%%%%%%%%%%%%%%%%%
```




- BAM/SAM, FASTA… will have longer life in the AS



23

# Table definition

```
<?xml version="1.0"?>
<Table>
<TableName>fastq</TableName>
<TableNumFields>5</TableNumFields>


<Field>
<FieldName>ID</FieldName>
<FieldLength>8</FieldLength>
<FieldOrdinal>0</FieldOrdinal>
<FieldNullBit>0</FieldNullBit>
<FieldType>LONGLONG</FieldType>
</Field>
<Field>
<FieldName>header</FieldName>
<FieldLength>100</FieldLength>
<FieldOrdinal>1</FieldOrdinal>
<FieldNullBit>0</FieldNullBit>
<FieldType>STRING</FieldType>
</Field>
<Field>
<FieldName>sequence</FieldName>
<FieldLength>100</FieldLength>
<FieldOrdinal>2</FieldOrdinal>
<FieldNullBit>0</FieldNullBit>
<FieldType>STRING</FieldType>
</Field>
<Field>
<FieldName>splitter</FieldName>
<FieldLength>100</FieldLength>
<FieldOrdinal>3</FieldOrdinal>
<FieldNullBit>0</FieldNullBit>
<FieldType>STRING</FieldType>
</Field>
<Field>
<FieldName>qualities</FieldName>
<FieldLength>100</FieldLength>
<FieldOrdinal>4</FieldOrdinal>
<FieldNullBit>0</FieldNullBit>
<FieldType>STRING</FieldType>
</Field>

<Key>
      <KeyParts>1</KeyParts>
      <KeyPart>
            <FieldOrdinal>0</FieldOrdinal>
      </KeyPart>
</Key>

</Table>
```


BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación
PORTING THE WORKFLOW

# Conversion Approach (1)

- The software components developed are:
  - **PIMDtools** is a command line tool that allow for the creation and manipulation of a FMIndex data structure residing on PIMD. The tool allows for the creation of a Suffix Array of a given input text, from that derives the BWT structure for the given input, and finally generates the C and Occ arrays defined by the FMIndex. The tool also allows for the process of loading FASTA and FASTQ files into a PIMD repository.
  - **Modified version of the BWA** tool that employs PIMD as the storage layer instead of plain files. The semantics of the tools have not been modified and the results are compatible with the vanilla version. The interaction with PIMD has been implemented through a set of external classes that provide the high-level interfaces needed to keep the BWA semantics internally. The modified methods involve the single and paired ended short read alignments.
  - **Emulation of SAMtools** interface (to some extent) using TupleStash and PIMDtools

# Conversion approach (2)

- Load data (arriving in disks)
  - If FASTA: build index of a reference genome
  - If FASTQ: alignment to a reference genome
- Build Index:
  - This process is not frequent for alignment, but is time consuming
  - This process will be used in the novo assembly, so it is good to have it implemented
- Alignment:
  - Align reads to the reference genome
  - State of the art: FMIndex (BWA, Bowtie)
- Data manipulation:
  - Sort, merge, create pileups, project on reference regions,…



27

# FM-Index construction on PIMD: SA -> BWT -> FMIndex

- BWT built from the Suffix Array of the sequence

- Parallel building of the Suffix Array and BWT (short description):
  - Load into PIMD using fixed size blocks (currently 2Kbps)
    - Key: Initial Offset; Value: 2KBps
  - Break sequence into k-mers (k=11 in our experiments) and store reference to occurrences of the k-mers in a PDS – done in parallel
    - k-mers are prefixes of the suffixes, they are stored as a hash value (lexicographically order kept)
    - Different processes go to different keys (with access to adjacent keys for trailing bases)
    - Store made using atomic operations (one-sided)
  - Sort k-mers and sort occurrences of each k-mer in parallel
    - Sort of occurrences of one k-mer is done accessing the sequence in parallel
    - Allows for very high-level of concurrency (as much as PIMD+network capacity allows)
  - Traverse the suffix array and produce BWT
    - stored in PIMD like the sequence: Key: Initial Offset; Value: 2KBps

Reference for FM-Index:

*P. Ferragina and G. Manzini. Opportunistic data structures with applications. In Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS '00). IEEE Computer Society, Washington, DC, USA*



29

# Storing the reference

- Parallel pimd insert using value offsets (atomic):
  - not overlapping
- Read-only after created
- Simple macros for:
  - position -> key
  - position -> offset  in value

Reference record

| Global Offset<br>mod 2k = 0 | Array of bases (2048 now, except last record) |
|---|---|



30

# FM-Index construction on PIMD - Runing example:

- FASTA File (12bps, |Σ| = 5, 2 records): ACCGNTTAACCT

>ENSDART00000008663 cdna:known chromosome:Zv9:25:18278:28255:1 gene:ENSDARG00000015502 …
ACCGNT
0 0 0 0 0 0
0 1 2 3 4 5
>ENSDART00000008663 cdna:known chromosome:Zv9:25:18278:28255:1 gene:ENSDARG00000015502 …
TAACCT
0 0 0 0 1 1
6 7 8 9 0 1

- Suffix array:

| SAI | Pos | Suffix |
|---|---|---|
| 01 | 07 | AACCT |
| 02 | 00 | ACCGNTTAACCT |
| 03 | 08 | ACCT |
| 04 | 01 | CCGNTTAACCT |
| 05 | 09 | CCT |
| 06 | 02 | CGNTTAACCT |
| 07 | 10 | CT |
| 08 | 03 | GNTTAACCT |
| 09 | 04 | NTTAACCT |
| 10 | 11 | T |
| 11 | 06 | TAACCT |
| 12 | 05 | TTAACCT |


31

# FM-Index construction on PIMD - Runing example:

- Reference: ACCGNTTAACCT
- Suffixes: AABBT
- T, CT, CCT, ACCT, …

- Prefixes (k-mers, k=3) of suffixes, totally ordered (ignoring shorter T, CT here):

AAC: 1
ACC: 2
CCG: 1
CCT: 1
CGN: 1
GNT: 1
NTT: 1
TAA: 1
TTA: 1

| Bucket0 | Bucket1 |
|---|---|
| 001 - AAC: 07<br>006 - ACC: 00,08<br>032 - CCG: 01<br>034 - CCT: 09<br>038 - CGN: 02 | 069 - GNT: 03<br>099 - NTT: 04<br>100 - TAA: 06<br>120 - TTA: 05 |

Alphabet mapping: A:0, C:1, G:2, N:3, T:4
libCDS: O(1) rank operations

Key: $\sum_{i=0}^{len-1} prefix[i] \cdot \|\Sigma\|^{len-i-1}$

Value: index in the sequence

N bins – N processors
Index rank: 0 (AAA) – 124 (TTT)
Key / N: 62
Sample Partitioner Hash function: floor(Key / (Key/N))



32

# FM-Index construction on PIMD - Runing example:

- Local Sorting:
  - Order between k-mers trivial -> stored in order by PIMD
  - Order of occurrences of 1 k-mer need access to ref.
  - The longer k is, the less collisions
  - Order of keys in a

| Bucket0 | Bucket1 | Trivial |
|---|---|---|
| 001 - AAC: 07<br>**006 - ACC: 00,08**<br>032 - CCG: 01<br>034 - CCT: 09<br>038 - CGN: 02 | 069 - GNT: 03<br>099 - NTT: 04<br>100 - TAA: 06<br>120 - TTA: 05 | CT<br>T |

| SAI | Pos | Suffix |
|---|---|---|
| 01 | 07 | AACCT |
| **02** | **00** | **ACCGNTTAACCT** |
| **03** | **08** | **ACCT** |
| 04 | 01 | CCGNTTAACCT |
| 05 | 09 | CCT |
| 06 | 02 | CGNTTAACCT |
| 07 | 10 | CT |
| 08 | 03 | GNTTAACCT |
| 09 | 04 | NTTAACCT |
| 10 | 11 | T |
| 11 | 06 | TAACCT |
| 12 | 05 | TTAACCT |



33

# Operations provided on the BWT: store

```
#define BPS_PER_RECORD_MASK   0xFFFFFFFFFFFF800
#define BPS_RECORD_SIZE_ALIGN(a) (uint64_t)((uint64_t)a & (BPS_PER_RECORD_MASK))

void BWT::store (                char *data, uint64_t Len, uint64_t position,
                                 pimd_client_cmd_ext_hdl_t *handlerOld, pimd_client_cmd_ext_hdl_t *handlerNew) {


        assert(Len <= BPS_PER_FASTA_RECORD)

        Key     = BPS_RECORD_SIZE_ALIGN(position);
        if(handlerOld!=NULL) Client.Wait( *handlerOld );   → Double buffering

         status = Client.iInsert( &m_BWT_PDS,
                          (char*)&Key,
                          sizeof(uint64_t),
                          data,
                          Len,
                          0,
                          (pimd_cmd_RIU_flags_t)( PIMD_COMMAND_RIU_INSERT_OVERLAPPING |
                                                  PIMD_COMMAND_RIU_INSERT_EXPANDS_VALUE ),
                          handlerNew);

}
```

# Operations provided on the FMIndex: rank (Occ access)

cIn: Character to search
Position: BWT position
Result: Number of

```
uint64_t FMIndexPIMD::rank(char cIn, int64_t position) {

        openBWT();

        status = Client.iRetrieve(&m_OCC_PDS,
                        (char *)&Key,
                        sizeof(uint64_t),
                        (char *)&cCount,
                        sizeof(cCount),
                        &size,
                        c*sizeof(uint64_t),
                        (pimd_cmd_RIU_flags_t)PIMD_COMMAND_RIU_FLAGS_NONE,
                        &handlerOCC);

        bwt->fetchBWTData(Value, readCount, Key, readBps);  → Accesses 'readBps/BLOCKSIZE' records in the BWT PDS to read 'readBps' bases

        Client.Wait(handlerOCC);

        uint64_t initial = cCount;

        for(uint64_t i = 0; i< readCount; i++)
           if(MAPCHAR3(Value[i]) == c)
                cCount++;

        return cCount;

}
```

## OCC_PDS structure

Key 0, Value: (0,0,0,0), count of (A,C,T,G) in BWT up to position 0
Key 2048, Value: (1024,769,215,40), count of (A,C,T,G) in BWT up to position 2048
…

# TupleStash Example: Store a FASTQ Record

```
void FASTQ::storeRecord(uint64_t id, char **fields, uint32_t *lens) {

    for(int i=0; i < 4; i++)
        rowBufferSize += lens[i]*2;

    char rowBuffer[rowBufferSize];

    fieldIndex=0; // ID
    m_tStash.SetRecordFieldLength(fieldIndex, sizeof(id));
    m_tStash.SetRecordFieldPointer(fieldIndex, (char *)&id);

    for(int i=0; i < 4; i++){
        fieldIndex=i+1;
        m_tStash.SetRecordFieldLength(fieldIndex, lens[i]);
        m_tStash.SetRecordFieldPointer(fieldIndex, (char *)fields[i]);
    }

    m_tStash.PackRecordToRow(rowBuffer, rowBufferSize);
    m_tStash.GetKeyValRefs(rowBuffer, &Key, KeyLen, &Value, ValueLen);

     pimd_status_t status = Client.Insert(   &m_FASTQ_PDS,
                                             Key,
                                             KeyLen,
                                             Value,
                                             ValueLen,
                                             0,
                                             (pimd_cmd_RIU_flags_t)PIMD_COMMAND_RIU_UPDATE
    );
```

# TupleStash Example: Traverse FASTQ Records in order

```
int FASTQ::getNextRecord(kseq_t **output) {

 if(!m_CursorOpen) {
            status = Client.CreateIndex(&m_FASTQ_PDS,
                        &m_tStash,
                        orderFields,
                        numOrderFields,
                        (relational_postfix_token_manager_t *)NULL,
                        (pimd_index_flags_t)(PIMD_INDEX_FLAGS_HASH_ORDERED | PIMD_INDEX_FLAGS_SHARED),
                        &m_IndexHandle
            );
             status = Client.CreateCursor(m_IndexHandle,
                        NULL,
                        numProjectedFields,
                        (relational_postfix_token_manager_t *)NULL,
                        (pimd_cursor_flags_t)PIMD_CURSOR_USE_SHARED_STREAM_FLAG,
                        8192,
                        &m_Cursor
            );

            m_CursorOpen = true;
 }

  status = Client.GetNextElement ( m_Cursor,
                        (char *)Value,
                        &ValueSize,
                        FASTQMaxValueSize,
                        (pimd_cursor_flags_t) PIMD_CURSOR_USE_SHARED_STREAM_FLAG);


     ** PROCESS CURRENT ELEMENT HERE **

}
```

# BWA conversion (1)

- Conversion straightforward once the index is created
- Process: replace access to index with PIMDtools calls
- Some additional metadata structures created to keep consistency between parallel BWA processes (storing BWA options and sharing across processes…)

***ACCESS TO FASTA/FASTQ SEQUENCE***

```
c = pacseq[(x+z)>>2] >> ((~(x+z)&3)<<1) & 3;
```

**becomes…**

```
c = sequence->fetchSequenceCharMapped(x+z);
```

***MAJOR ALIGNMENT LOOP (bwa_aln_core):***

- No need to read FMIndex
- Only open FMIndex PDS's (BWT, C, Occ...) and FASTA + FASTQ
- Get FASTQ reads on a work-stealing approax (FCFS, atomic operation)
- Process alignment (replacing access to compressed FMIndex with calls to PIMD data) as shown later
- Store alignment (atomic operation) to Alignments PDS

*alignments->storeAlignment(p->readID, aln_id, aln->k, aln->l, aln->score, aln->n_mm, aln->n_gape, aln->n_gapo);*



38

# BWA conversion (2)

## Alignment refinement, single end reads '*samse*' (bwt_cal_width ):

```
PIMDTools::FMIndexPIMD *fmi = new PIMDTools::FMIndexPIMD((char *)prefix, 1, true, tsDefFASTAmetadata);

    k = 0; l = fmi->getSequenceSize();
    for (i = 0; i < len; ++i) {
        ubyte_t c = str[i];
        if (c < 4) {
            //bwt_2occ(bwt, k - 1, l, c, &ok, &ol);  k = bwt->L2[c] + ok + 1;  l = bwt->L2[c] + ol;  ORIGINAL BWA CODE
            ok=fmi->rank(c,k-1, true);    ol=fmi->rank(c,l, true);
            k = fmi->getC(c, true) + ok + 1;  l = fmi->getC(c, true) + ol;
        }
        if (k > l || c > 3) { // then restart
            k = 0;
            //l = bwt->seq_len;  ORIGINAL BWA CODE
            l = fmi->getSequenceSize();
            ++bid;
        }
        width[i].w = l - k + 1;
        width[i].bid = bid;
    }
```



39