Bio Data Management

Kapitel 5b

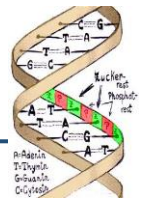
NGS Data Management

Wintersemester 2014/15
Dr. Anika Groß
Universität Leipzig, Institut für Informatik, Abteilung Datenbanken
http://dbs.uni-leipzig.de

UNIVERSITÄT LEIPZIG

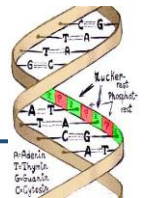
Vorläufiges Inhaltsverzeichnis

1. Motivation und Grundlagen
2. Bio-Datenbanken
3. Datenmodelle und Anfragesprachen
4. Modellierung von Bio-Datenbanken
5. a) Sequenzierung und Alignments
   b) NGS Data Management
6. Genexpressionsanalyse
7. Annotationen
8. Matching
9. Datenintegration: Ansätze und Systeme
10. Versionierung von Datenbeständen
11. Neue Ansätze



Lernziele und Gliederung

- Management von Next Generation Sequencing (NGS) Daten
  - Bestimmung des Read Mapping,
    Cloud-basierte Verfahren (CloudBurst)
  - Verfahren zur Sequenzkompression
  - Parallelisierung von Scientific Workflows



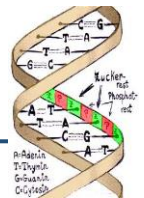
Sequenzierung

- Früher: limitiertes Level an Parallelisierung
  - Parallele Verarbeitung von 40-100 reads
  - Hohe Genauigkeit von 99.999% pro Base
  - Im Bereich der high-throughput shotgun Sequenzierung kostet die Sanger Sequenzierung ca. $0.50 pro Kilobase

→ **Next Generation Sequencing**

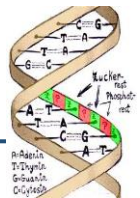
Literatur:

- Wandelt, Rheinländer, Bux, Thalheim, Haldemann, Leser: Data Management Challenges in Next Generation Sequencing. *Datenbank-Spektrum*, 2012.
- Shendure, Ji: Next-generation DNA sequencing. *Nature biotechnology*, 2008.



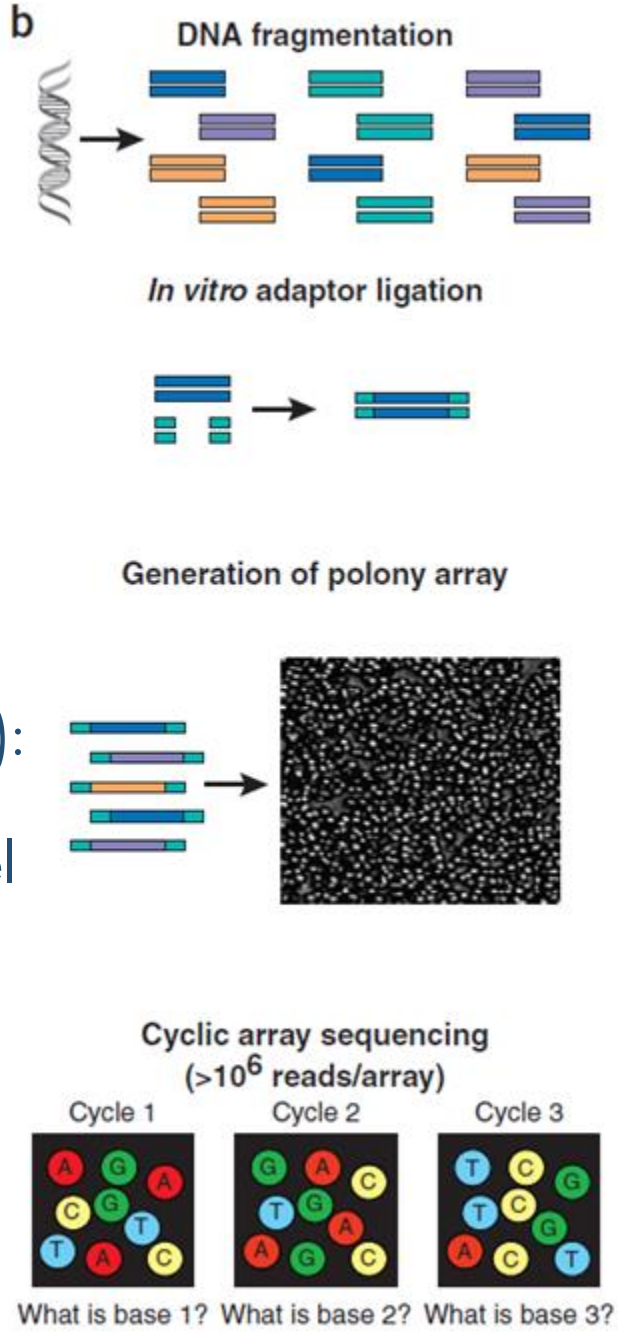
Next Generation Sequencing (NGS)

- Parallele Sequenzierung von Millionen - Milliarden von reads
- Produzieren mehrere Terabytes „raw signal“
  → mehrere 100 GB Sequenzdaten in einer Woche, auf einer Maschine
- Sequenzierung ist Routine, kaum mehr Forschung
- Anwendungen: Personalisierte und translationale Medizin (z.B. neue Diagnosemöglichkeiten mit NGS), zielgerichtete Untersuchung von Mutationen und Polymorphismen, …
- Verschiedene Methoden:
  z.B. „cyclic-array sequencing“ (454 Genome sequencer von Roche, Illumina Genome Analyzer von Solexa, …)
- *„Sequencing of a dense array of DNA features by iterative cycles of enzymatic manipulation and imaging-based data collection“*
  (Shendure and Ji, 2008)



Cyclic-array Sequencing

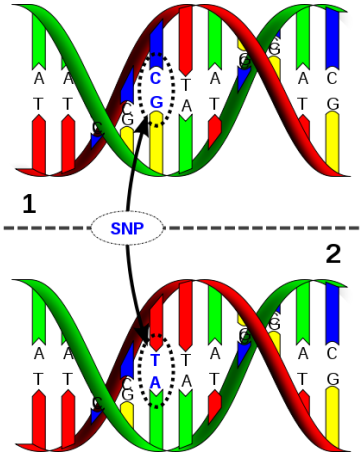
- Zufällige Fragmentierung der DNA (*library*)
- *in vitro* Ligation (enzymatisch gesteuerte Verknüpfung/Bindung) von gemeinsamen Adaptersequenzen
- Amplifikation (Vermehrung) der Fragmente durch z.B. PCR (*polymerase chain reaction*)
  → over-sampling, um Abdeckung der gesamten Sequenz zu gewährleisten
- Array von Millionen von PCR Kolonien (befestigt auf Platte), jede Kolonie (auch Cluster): viele Kopien eines Fragments
- Sequenzieren mehrere Millionen Cluster parallel + z.B. bei Solexa 8 Libraries parallel
- Sukzessive Iteration (Zyklen):
  - Enzymatisches Reagenz auf Array, Erweiterung um einzelne Base pro Zyklus
  - Aufnahme eines Bildes pro Zyklus (Erkennung der 4 Floureszenzmarkierungen)



1000 Genomes Project

http://www.1000genomes.org/

- Sequenzierung der Genome von rund 2500 Menschen
- Start 2008, Institute zahlreicher Länder:
  USA, England, China, Deutschland, …
- Detaillierter Katalog menschlicher genetischer Variationen, die in mind. 1% der untersuchten Populationen vorkommen
  (z.B. Single Nucleotide Polymorphismen (SNP), INDELs, strukturelle Variationen wie Kopienzahlvariationen)
- Komplette Datenbank wird Wissenschaftlern weltweit kostenlos zur Verfügung gestellt

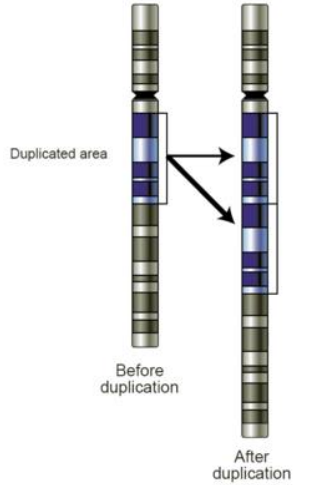


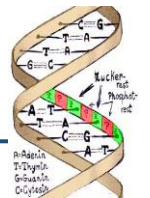
Weiteres Projekt: UK10K (seit 2011)

- Untersuchung von ~10.000 humanen Genomen, um seltene Varianten verschiedener Krankheitstypen zu untersuchen
- Finden von Assoziationen zwischen genetischen Variationen und phänotypischen Eigenschaften

UK 10K

http://www.uk10k.org/





„Das 1000$ Genom“

- Ziel der Biotechbranche: Genomsequenzierung als Routinetest
- Bisher optische Sequenzierautomaten z.B. von Illumina für 500.000$

heise online

ct · iX · Technology Review · Mac & i · mobil · Security
Download · Telepolis · Resale · Foto · Autos · Preisver

Technology Review

Forum Archiv

Sie sind Gast · Einloggen | Registrieren

Energie | Infotech | Leben | Materie | Prod

Technology Review > Leben > Das 1000-Dollar-Genom

Das 1000-Dollar-Genom

16.01.12 – Erica Westly

Schlagwörter: Genchip, Gensequenzierung

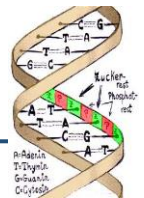
Die Biotechfirma Ion Torrent hat einen Preisbrecher im Markt der Gen-Sequenzierautomaten vorgestellt.

Die Sequenzierung des gesamten menschlichen Genoms war lange ein Privileg für Menschen mit genügend Kleingeld. Die US-Biotechfirma Ion Torrent will den Job künftig zum Preis von knapp 1000 US-Dollar pro Durchgang erledigen. Möglich macht dies ein neuer Sequenzierautomat, den Firmenchef Jonathan Rothberg am Rande der CES in Las Vegas vorstellte.

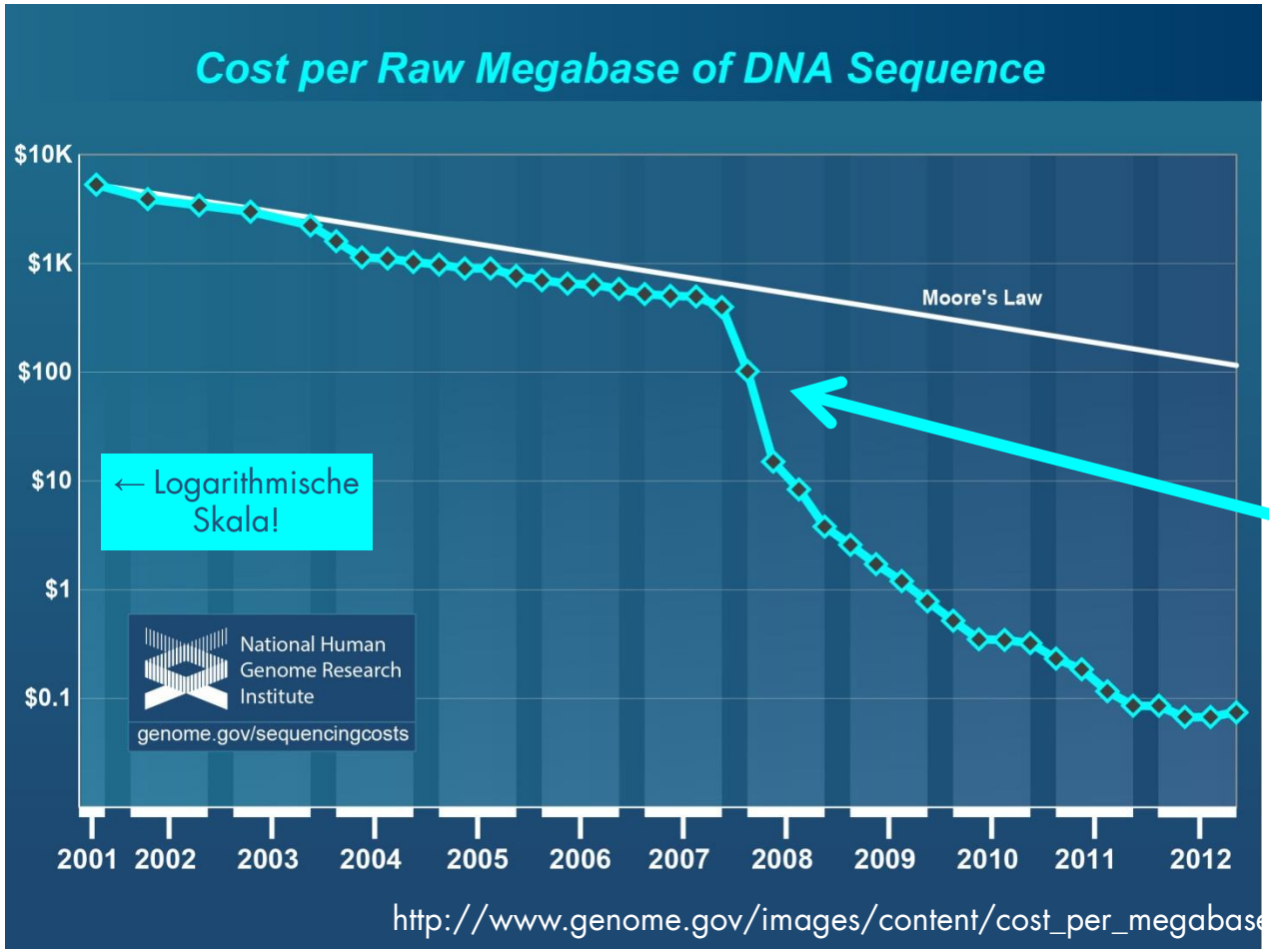
http://www.heise.de/tr/artikel/Das-1000-Dollar-Genom-1413228.html

- „Ion Proton“ passt auf Tischplatte
- vorerst nur für Wissenschaftler, bald auch auf regulären Medizintechnikmarkt
- 149.000$
- DNA-Chip ist 1.000 Mal leistungsfähiger (bzgl. Vorgänger)
- Sequenzierung des gesamten menschlichen Genoms in nur einem Tag für 1.000$!
- Alternativ Exom-Analyse für 500$ (nur Exons)

… bald 100$ Genom …



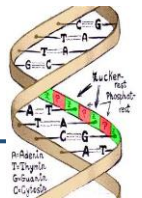
Kosten DNA Sequenzierung



Übergang von Sanger-basierten Verfahren auf „Next Generation DNA" Sequenzierung

Kosten pro Megabase
$5.292,39 (2001)
→ $0,07 (2012)

- Das „Moore'sche Gesetz" besagt, dass sich die Leistungsfähigkeit von Prozessoren etwa alle 1-2 Jahre verdoppelt.
- Gilt auch für die Energieeffizienz von Rechnern („Koomey'sches Gesetz")



NGS - Vorteile/Nachteile

Vorteil

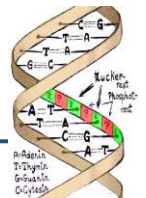
- besser parallelisierbar, weil
  - *in vitro* statt *in vivo*
  - Array-basiert statt kapillarbasiert
  - Befestigung der Fragmente auf einer Platte, enzymatische Manipulation mit einem Reagenz

Nachteil

- Niedrigere Qualität: Read-Länge sehr kurz, 10 mal weniger akkurat als klassische Sanger-Sequenzierung

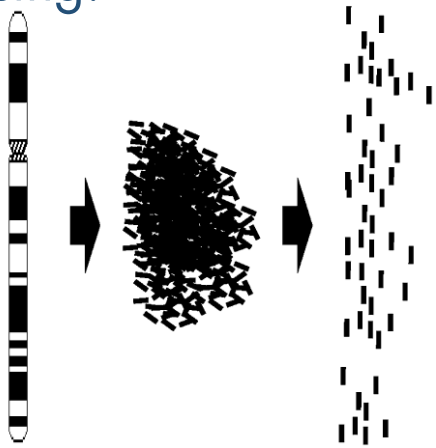
- Mappen Millionen von „short reads“ gegen ein Referenzgenom (statt Assembly)

→ Benötigen fortgeschrittene Datenstrukturen + hochparallele Algorithmen

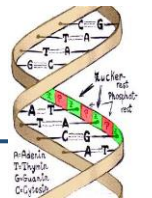


Data Management Challenges in NGS

Literatur: Wandelt, Rheinländer, Bux, Thalheim, Haldemann, Leser: Data Management Challenges in Next Generation Sequencing. *Datenbank-Spektrum*, 2012.



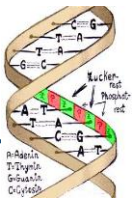
1) Bestimmung des Read Alignments/Mappings
   - Viele kurze NGS ‚reads' müssen gegen Referenzsequenz (z.B. humanes Genom) gemappt werden → erfordert effiziente Lösungen!
2) Sequenzkompression
   - Speicherung und Übertragung von Genomdaten wird problematisch durch NGS
     - 1000 Genome → 3 Terabyte; noch wesentlich mehr mit Metadaten, Qualitätsinformationen etc.
     - Limitierungen der Bandbreite behindern den Datentransfer zwischen versch. Infrastrukturen, z.B. für übergreifende Analysen
3) Scientific Workflows
   - Gleichzeitige Analyse von 100en Genomen
   - Komplexe Pipelines zahlreicher Tools



Read Mapping (Read Alignment)

- Mapping der '*reads*' zu einer Referenzsequenz (Referenzgenom)
- Zweck: finde alle „fast exakten" Matches der reads in der bekannten Referenzsequenz
  - Für jedes gegebene $read\ r$
    - Finde alle 4-Tupel (c, s, e, p)
    - c - Chromosom, s - pos/neg Strang,
      p - relative Startposition von r in c, e - Fehler
    - Anzahl erlaubter Fehler: 1-10% , Fehler-Art: nur Mismatches <u>oder</u> Mismatches + Indels (abh. von Anwendung)
- Hohe Datenmenge durch NGS-Geräte → brauchen Algorithmen zur Bestimmung des Read Mapping mit sub-linearer Laufzeit
  1) Indexierung der Referenzsequenz
     (z.B: Suffixbäume, Hashing von q-Gramen)
  2) „seed-and-extend" Algorithmen sind schnell,
     denn sie lösen das Problem nur „ausreichend" gut

→ Nutze Index, um schnell kurze exakte Übereinstimmungen zu finden <u>und verzichte auf „optimales Alignment" (niedrigere Qualität)</u>

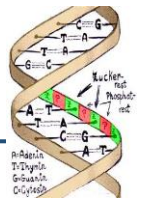


„seed-and-extend“ Algorithmen

- **Idee:** In jedem Alignment mit wenig Fehlern gibt es einen Bereich, in welchem beide Sequenzen exakt übereinstimmen.
- Mindestanzahl korrekter, aufeinanderfolgender Basen gefordert: *seed* (dt.: Saat, Same, Keim)

Ein Aligment, das maximal e Mismatches erlaubt, muss mind. einen exakten Match (seed) der Länge $\frac{|r|}{e+1}$ zu einem Substring des reads r enthalten.

- Beispiel: 30 bp read
  - 1 Fehler: seed-Länge = 15
  - 2 Fehler: seed-Länge = 10



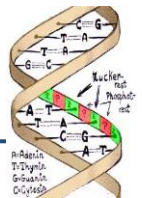
„seed-and-extend“ Algorithmen

- Finden der *seeds* durch Vergleichen der reads gegen eine indexierte Referenzsequenz
- *extend:* Verlängern der *seeds* an beiden Seiten
  → Finden des vollständigen Alignments

- Regionen ohne seeds werden ohne weitere Betrachtung gefiltert

- Bsp. BLAST:
  - *seed*: HashTable aller k-mere fester Länge in der Referenzsequenz zum Finden der seeds
  - *extend*: Smith-Waterman (Band Variante) zur Berechnung von „high-score, gapped Alignment“

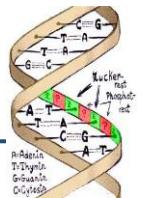
- *seed-and-extend* Methode gut geeignet für kurze reads (<100bp), da diese selten viele Fehler oder große Lücken enthalten



Aber ...

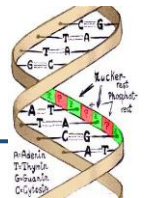
… NGS Methoden verbessern sich

- reads werden länger „long reads“
  → fehleranfälliger (mehr Indels), d.h. Algorithmus für „long reads“ sollte mit größeren Lücken umgehen können
- „short read“ Mapping Tools: nur effizient für lückenlose o. lückenarme Alignments
- BLAST gut geeignet für long read mapping,
  aber nur bei wenig reads (skaliert nicht für NGS)
- Long read alignment: verschiedene Methoden,
  z.B. speichereffiziente Genom-Indices, Multi-threaded execution, gapped seeds



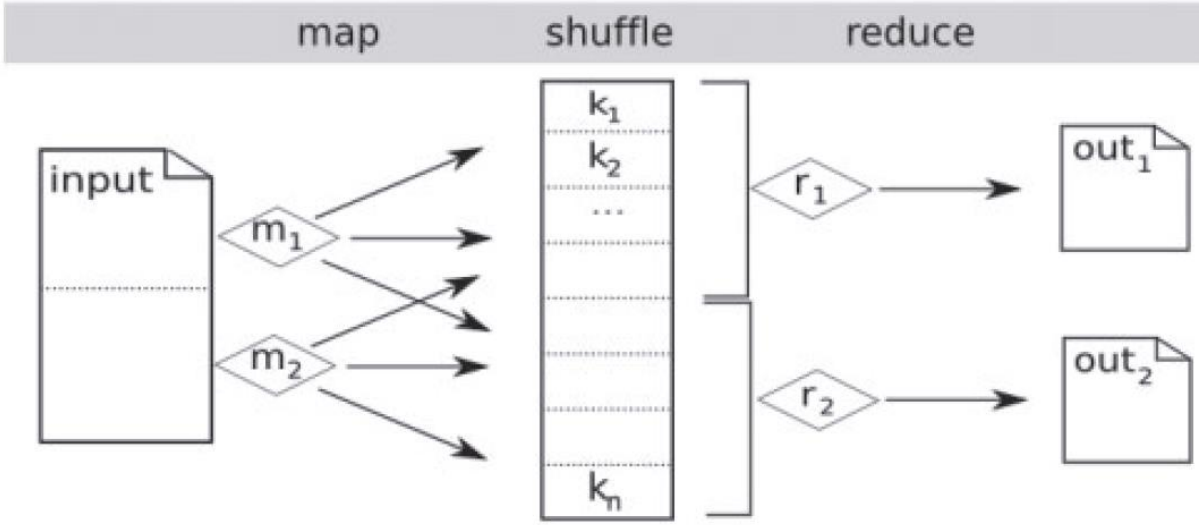
Cloud-basierte Lösungen

- Zur weiteren Beschleunigung der Read Mapping Bestimmung: Parallelisierung
- Nutzen von *MapReduce* für parallele Ausführung
  - Effektives, robustes Management der Parallelisierung
  - Verteilung der Inputdaten und des User-Quellcodes auf hohe Anzahl an computing nodes
- *seed-and-extend*: sehr gut parallelisierbar
  - Jedes read kann individuell/unabhängig gegen Referenzgenom gemapped werden (keine Abhängigkeiten zu anderen reads)
- Beispiele: CloudBurst, Crossbow, SeqMapReduce, CloudAligner, …



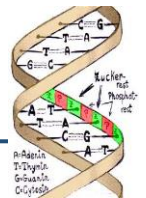
MapReduce

- Programmiermodell zur parallelen Ausführung datenintensiver Berechnungen
- von Google; freie Implementierung von Hadoop
- zwei Hauptphasen: „map“, „reduce“
  + interne, optionale „shuffle“ Phase



MapReduce

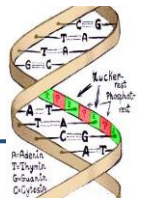
- map
  - Berechnet key-value Paare aus den Inputdaten
  - Viele Instanzen der map-Funktion arbeiten parallel auf Inputdaten
- shuffle
  - Gleiche Funktionalität wie die Reduce-Funktion, wird aber auf dem gleichen Knoten wie die Map-Phase ausgeführt (Reduktion der Netzwerklast)
  - Alle Werte mit gleichem key gruppieren
    → HashTable, mit key auf Liste von Werten
- reduce
  - Durch Nutzer definierte Funktion
  - z.B. Summe über Werte in Liste



Probleme / Limitierungen

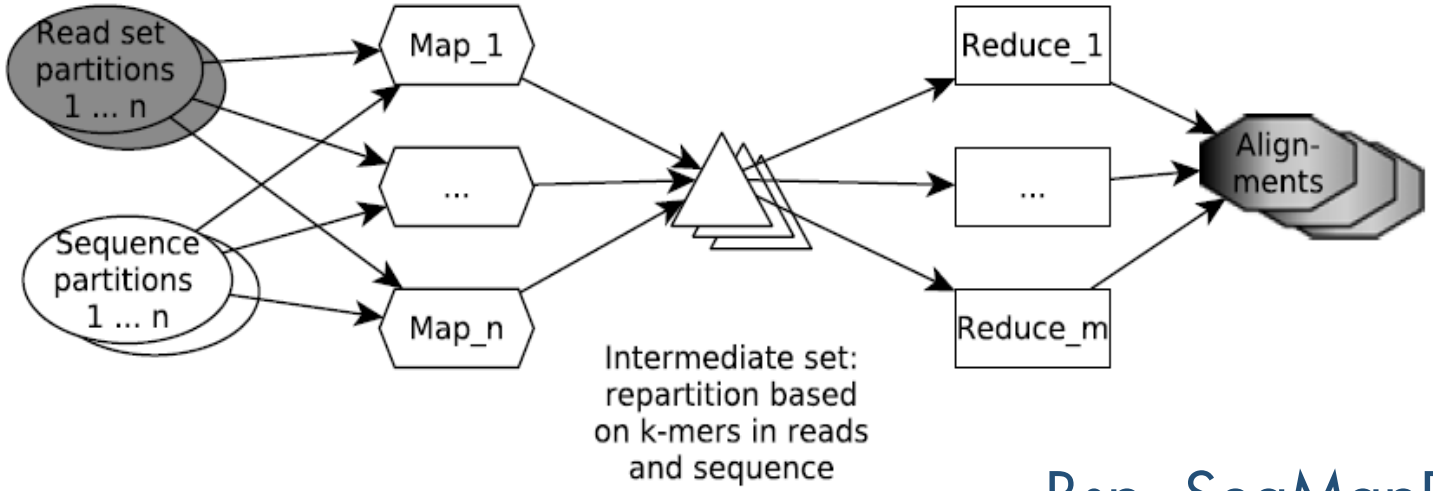
- Ziel: Linearer speedup bzgl. Anzahl verfügbarer Prozessor-Cores
- Meist wird kein linearer speedup erreicht
  - Sequenzielle Anteile des Programms,
    nicht parallelisierbarer Overhead → bestimmt Mindestlaufzeit
  - Limitierung durch langsamsten Task: zusätzliche Lastbalancierung nötig, wenn einfache Partitionierung (z.B. 1/#cores) nicht ausreicht

- Sehr große Datensätze passen nicht in Hauptspeicher
  - Dateien zum Speichern und Übertragen von Zwischenergebnissen
    - Google File System (GFS), Hadoop Distributed File System (HDFS)
    - Hohe Bandbreite für MapReduce durch Partitionierung und Replikation der Dateien über mehrere DataNodes
  - MapReduce ist „data-aware“: Berechnung wird an Knoten geschickt, der die Daten vorrätig hat, statt Daten im Netzwerk zu verschieben
  - Abhängigkeit von Bandbreite des Cloud Anbieters
    z.B. für NGS: Zeit für initialen Datentransfer ist sehr hoch

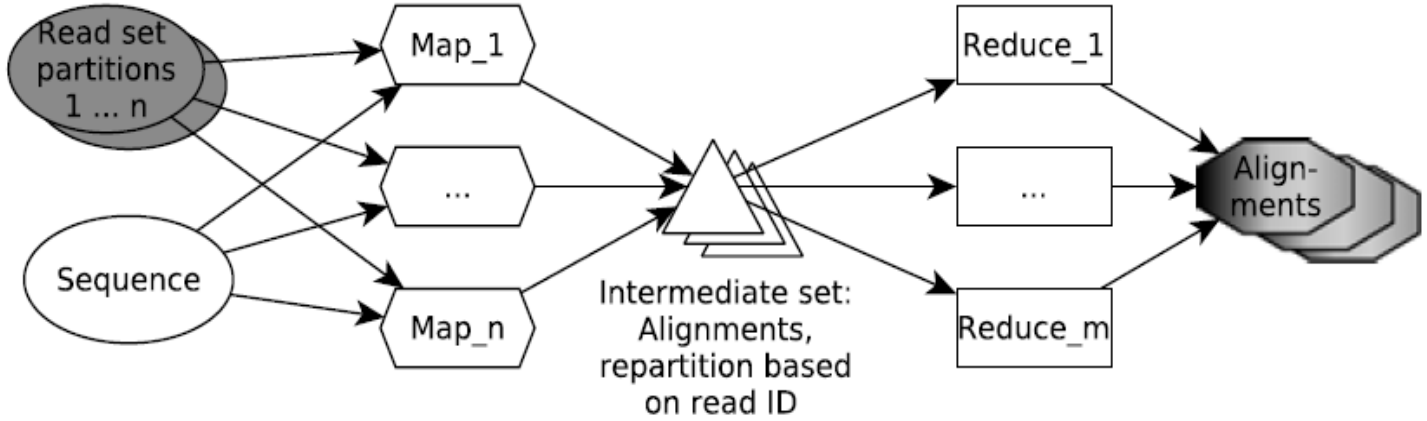


Partitionierung – Read Alignment Tools

Bsp. CloudBurst



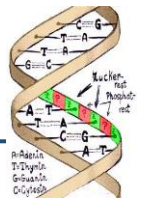
Bsp. SeqMapReduce



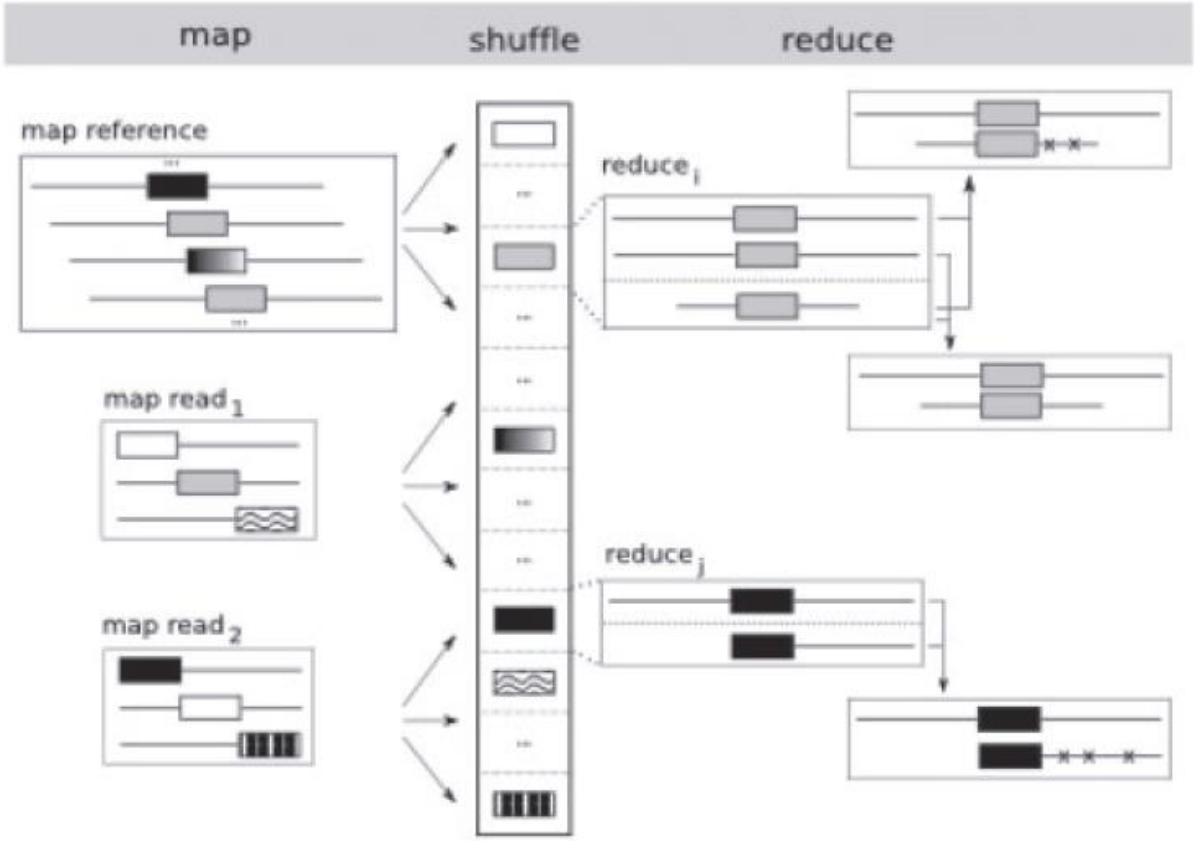
Wandelt, Rheinländer, Bux, Thalheim, Haldemann, Leser:
Data Management Challenges in Next Generation Sequencing. *Datenbank-Spektrum*, 2012.

CloudBurst

- MapReduce-basierter Read Mapping Algorithmus
- Ziel: viele kurze reads zum Referenzgenom mappen
- *seed-and-extend* wie bei Tool RMAP
- Input
  - 1 multi-FASTA-Datei mit reads
  - 1 multi-FASTA-Datei für ein oder mehrere Referenzsequenzen
  - DNA-Sequenz als *(key,value)*-Paare: *(id,SeqInfo)*
    SeqInfo-Tupel: *(sequence, start_offset)*
  - Kopieren in HDFS (Konvertierung: Binary Hadoop Sequence Files)
  - Referenzsequenzen aufgeteilt in je 65kbp große chunks
    (jeweils Überlappung von 1 kbp)
- Output:
  - Menge von Alignments: jedes Alignment für jedes *read* mit max. *e* Mismatches
  - In binary file, Konvertierung in tab-separiertes Standardformat möglich
    (z.B. wie RMAP Ausgabe)



CloudBurst - Überblick



Schatz, M. C.: CloudBurst: highly sensitive read mapping with MapReduce. *Bioinformatics*, 25(11), 2009

map

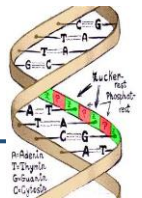
- Scan der Inputsequenzen und Ausgabe der *(key,value)*-Paare $(seed, MerInfo)$
  - *MerInfo-Tupel*: $(id, pos, isRef, isRC, leftFlank, rightFlank)$

    pos – Position/Offset des seed in Originalsequenz

    $isRef$– is reference Sequence? 0 oder 1

    $isRC$ – is reverse complement? 0 oder 1

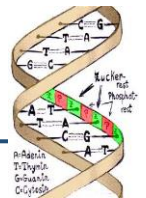
    „Merken" der flankierenden Sequenzen $leftFlank, rightFlank$: bis zu $(|r| - |seed| + e)$ Basenpaare
- Inputsequenz = Referenzsequenz R
  - Ausgabe (seed,MerInfo) für jede Position in der Referenz
  - $isRef = 1,\ isRC = 0$
  - Anzahl "Referenz-seeds": $|R| - |seed| + 1$



map

- Inputsequenz = read r
  - Splitten eines reads in $(e+1)$ Teile (nicht überlappende seeds)
  - ein (key,value)-Paar $(seed, MerInfo)$ pro seed der Länge $\frac{|r|}{e+1}$
  - $isRef = 0$
  - Zusätzlich alle *seeds als reverses Komplement* ($isRC = 1$), weil wir „read-Richtung" nicht kennen (wissen nicht, ob read in 3′→5′ oder 5′→3′ Richtung aus Sequenzierer kommt)
  - read-Länge konstant (z.B. immer Länge 36bp)

- Seeds: $\frac{2bit}{bp}$ Kodierung (A,C,G,T)
- Flankierende Sequenzen: $\frac{4bit}{bp}$
  (A,C,G,T, unbekannte Base N, Separatorsymbol .)



Beispiel

Reference sequence

ATGCAGTCATCGCAA

ATG TGC GCA CAG
AGT GTC TCA CAT
ATC TCG CGC GCA
CAA

Reads

AGTTAT

ATCGCA

AGT TAT

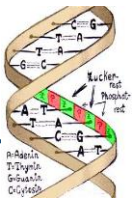
ATC GCA

$|r| = 6$

$e = 1$

$$|seed| = \frac{|r|}{e+1} = \frac{6}{1+1} = 3$$

Flankierung: bis zu $(|r| - |seed| + e) = 6 - 3 + 1 = 4$ Basenpaare



Beispiel - map

(seed, MerInfo)

$MerInfo: (id, pos, isRef, isRC, leftFlank, rightFlank)$

Reference sequence R1

ATGCAGTCATCGCAA

$(ATG, (R1,1,1,0, -, CAGT))$
$(TGC, (R1,2,1,0, A, AGTC))$
$(GCA, (R1,3,1,0, AT, GTCA))$
$(CAG, (R1,4,1,0, ATG, TCAT))$
$(AGT, (R1,5,1,0, ATGC, CATC))$
$(GTC, (R1,6,1,0, TGCA, ATCG))$
$(TCA, (R1,7,1,0, GCAG, TCGC))$
$(CAT, (R1,8,1,0, CAGT, CGCA))$
$(ATC, (R1,9,1,0, AGTC, GCAA))$
$(TCG, (R1,10,1,0, GTCA, CAA))$
$(CGC, (R1,11,1,0, TCAT, AA))$
$(GCA, (R1,12,1,0, CATC, A))$
$(CAA, (R1,13,1,0, ATCG, -))$

Reads re1 und re2

AGTTAT

ATCGCA

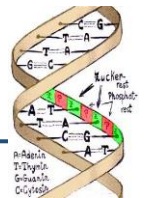
$(AGT, (re1,1,0,0, -, TAT))$
$(TAT, (re1,4,0,0, AGT, -))$
$(ATC, (re2,1,0,0, -, GCA))$
$(GCA, (re2,4,0,0, ATC, -))$

+als reverses Komplement

ATAACT

TGCGAT

$(ATA, (re1,1,0,1, -, ACT)$
$(ACT, (re1,4,0,1, ATA, -)$
$(TGC, (re2,1,0,1, -, GAT)$
$(GAT, (re2,4,0,1, TGC, -)$



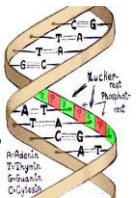
shuffle + reduce

shuffle

- Gruppieren gleicher seeds (aus Referenz und reads); haben gleichen key
- Output: Liste von MerInfos für jedes seed

reduce

- Erweitern (*extend*) der gemeinsamen seeds in längere (end-to-end), nicht-exakte Alignments zwischen rechter und linker Flankierung
- Pro seed: Partitioniere MerInfo-Liste in R Referenz-seeds und Q read-seeds
  $\rightarrow R \times Q$ Möglichkeiten: aligniere jedes Paar
  - Suchen alle Alignments mit maximal e Unterschieden
  - Scannen der flankierenden Sequenzen und Zählen der Mismatches (Lücken sind in CloudBurst nicht zulässig!)
  - Statt Mismatches Zählen: Nutzen eines DP-Alg. (Kap.5a) falls Lücken erlaubt sind (hier Landau-Vishkin k-Difference Algorithmus für gapped-Alignments)
  - Entferne Duplikate: es gibt $(e + 1)$ seeds pro read
    $\rightarrow$ finden gleiches Alignment mehrmals
- Blockweise über Submengen von R und $Q \rightarrow$ guter Cache Reuse
- Nutzen der flankierenden Basen gemeinsamer seeds (aus $MerInfo$)

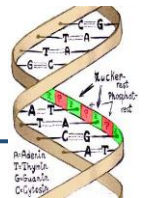


Beispiel - shuffle

$(ATG, (R1,1,1,0, -, CAGT))$
$(TGC, (R1,2,1,0, A, AGTC))$
$(GCA, (R1,3,1,0, AT, GTCA))$
$(CAG, (R1,4,1,0, ATG, TCAT))$
$(AGT, (R1,5,1,0, ATGC, CATC))$
$(GTC, (R1,6,1,0, TGCA, ATCG))$
$(TCA, (R1,7,1,0, GCAG, TCGC))$
$(CAT, (R1,8,1,0, CAGT, CGCA))$
$(ATC, (R1,9,1,0, AGTC, GCAA))$
$(TCG, (R1,10,1,0, GTCA, CAA))$
$(CGC, (R1,11,1,0, TCAT, AA))$
$(GCA, (R1,12,1,0, CATC, A))$
$(CAA, (R1,13,1,0, ATCG, -))$
$(AGT, (re1,1,0,0, -, TAT))$
$(TAT, (re1,4,0,0, AGT, -))$
$(ATC, (re2,1,0,0, -, GCA))$
$(GCA, (re2,4,0,0, ATC, -))$

$(AGT, (R1,5,1,0, ATGC, CATC))$
$(AGT, (re1,1,0,0, -, TAT))$
$(ATC, (R1,9,1,0, AGTC, GCAA))$
$(ATC, (re2,1,0,0, -, GCA))$
$(GCA, (R1,3,1,0, AT, GTCA))$
$(GCA, (R1,12,1,0, CATC, A))$
$(GCA, (re2,4,0,0, ATC, -))$
$(ATG, (R1,1,1,0, -, CAGT))$
$(TGC, (R1,2,1,0, A, AGTC))$
$(CAG, (R1,4,1,0, ATG, TCAT))$
$(GTC, (R1,6,1,0, TGCA, ATCG))$
$(TCA, (R1,7,1,0, GCAG, TCGC))$
$(CAT, (R1,8,1,0, CAGT, CGCA))$
$(TCG, (R1,10,1,0, GTCA, CAA))$
$(CGC, (R1,11,1,0, TCAT, AA))$
$(CAA, (R1,13,1,0, ATCG, -))$
$(TAT, (re1,4,0,0, AGT, -))$

$(AGT, (R1,5,1,0, ATGC, CATC), (re1,1,0,0, -, TAT))$
$(ATC, ((R1,9,1,0, AGTC, GCAA), (re2,1,0,0, -, GCA)))$
$(GCA, ((R1,3,1,0, AT, GTCA), (R1,12,1,0, CATC, A), (re2,4,0,0, ATC, -)))$



Beispiel - reduce

- Pro seed: Partitioniere MerInfo-Liste in R Referenz-seeds und Q read-seeds
- $R \times Q$ Möglichkeiten: aligniere jedes Paar
- Scanne flankierende Sequenzen und zähle Mismatches (+ggf. Lücken)

$(AGT, (R1,5,1,0, ATGC, CATC), (re1,1,0,0, -, TAT))$

R: $(R1,5,1,0, ATGC, CATC)$
Q: $(re1,1,0,0, -, TAT)$

ATGCAGTCATC
----AGTTAT-

1 Mismatch ✚

$(ATC, ((R1,9, 1,0, AGTC, GCAA), (re2,1,0,0, -, GCA)))$

R: $(R1,9, 1,0, AGTC, GCAA)$
Q: $(re2,1,0,0, -, GCA)$

AGTCATCGCAA
----ATCGCA-

0 Mismatches ✚

$(GCA, ((R1,3,1,0, AT, GTCA), (R1,12,1,0, CATC, A), (re2,4,0,0, ATC, -)))$

R: $(R1,3,1,0, AT, GTCA)$,
$(R1,12,1,0, CATC, A)$

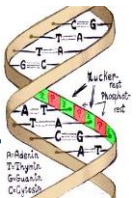
-ATGCAGTCA
ATCGCA----

3 Mismatches ✖
(1 Lücke)

Q: $(re2,4,0,0, ATC, -)$

CATCGCAA
-ATCGCA-

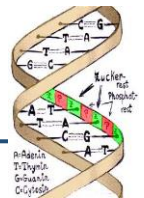
0 Mismatches ✚



Beispiel - Ergebnis

```
                123456789012345
                ATGCAGTCATCGCAA
R1-5,  re1-1        AGTTAT
R1-9,  re2-1             ATCGCA
R1-12, re2-4             ATCGCA   ✖
```

→ Duplikate entfernen
(entstehen durch $(e + 1)$ seeds pro read)

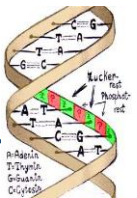


Lastbalancierung

- Parallelisierung pro seed, d.h. jeder Reducer führt alle potenziellen Alignments für ca. $^1/_N$ der $4^{|seed|}$ seeds aus $(N - |verfügbare\ Cores|)$
- Probleme bei *low-complexity/high-frequency seeds*, z.B. ‚AAAA' taucht überproportional oft auf → wesentlich längere Ausführungszeit der betroffenen Reducer notwendig
- „Rebalance"
  - Übergeben redundanter Kopien (z.B. ‚AAAA-1', ‚AAAA-2', ‚AAAA-3', ‚AAAA-4') in weitere Partitionen (pro Auftreten in der Referenzsequenz)
  - Die reads werden zufällig seed ‚AAAA-R' (mit $0 \leq R \leq 3$) zugeordnet

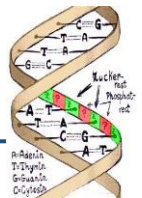
  → Gesamtanzahl durchgeführter Alignments bleibt gleicht

  → Aber parallele Verarbeitung auf mehreren Reducern



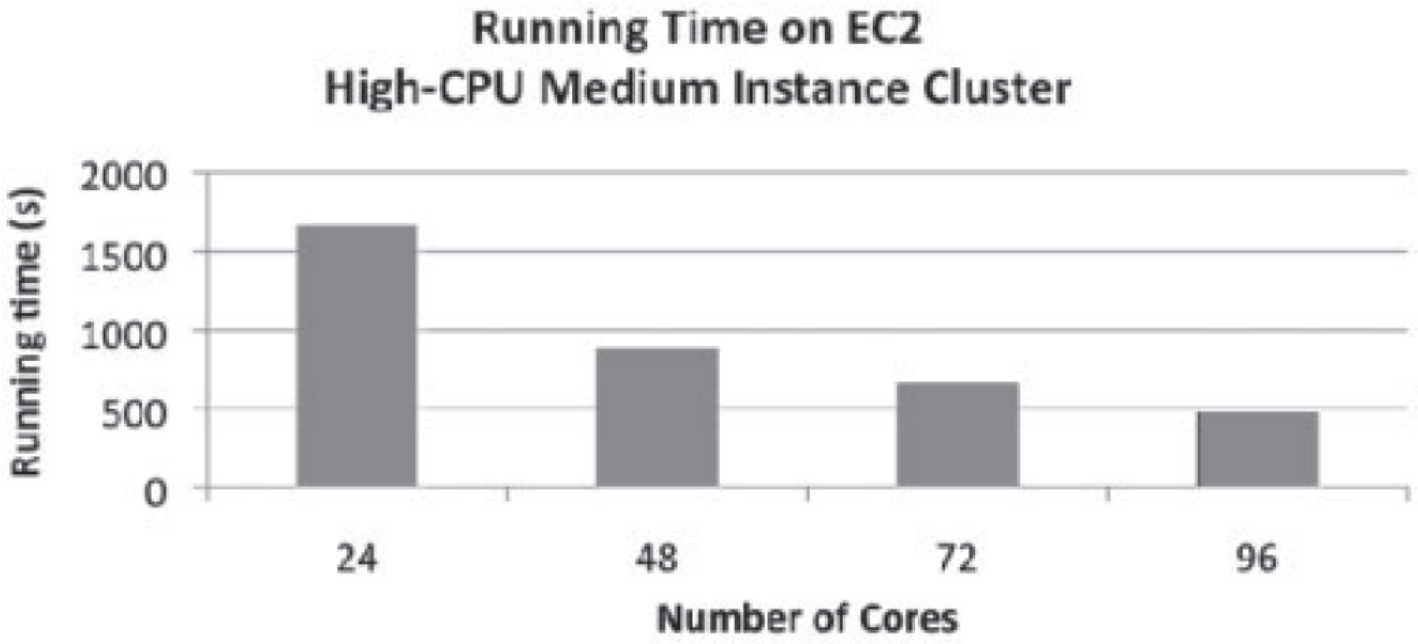
Filterung der besten Alignments

- Je nach Anwendung wird eventuell nur das beste Alignment pro read benötigt (statt aller Alignments)
- Filterung: nimm pro read jeweils das mit den wenigsten Mismatches/Unterschieden
- Falls ein read mehrere beste Alignments hat, dann wird kein Alignment ausgegeben (wie in RMAPM)
- Implementierung: 2. MapReduce Algorithmus:
  - Map: Übergabe aller ermittelten Alignments *(read-identifier,aligment-info)*
  - Shuffle: Gruppierung nach read
  - Reduce: Scannen der Alignments-Liste pro read und Auswahl des besten Alignment für read



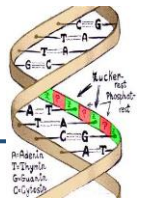
Experiment

- CloudBurst Laufzeit für Mappen von 7Millionen reads gegen Chromosom 22 ($e = 4$)
- Nutzen EC2Cluster: 96-core cluster ist 3.5× schneller als 24-core cluster



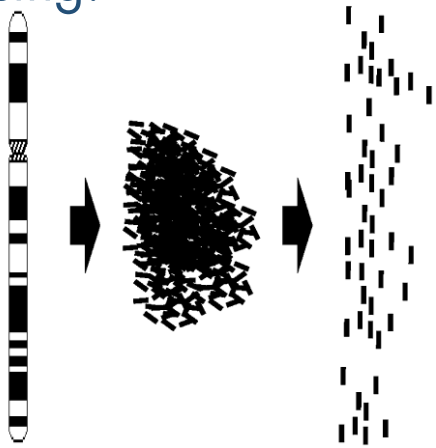
Offene Fragestellungen

- Skalierbarkeit: unklar, ob aktuelle verteilte Read Alignment Verfahren die großen Datenmengen in angemessener Zeit verarbeiten können
- Spezielle Algorithmen für große Lücken notwendig:
  z.B. mRNA „zurück alignieren“ auf DNA (Introns)
  → nicht mit zufriedenstellender Genauigkeit durch aktuelle heuristische Verfahren lösbar
- Integration von „quality scores“ in Read Alignment:
  Sequenzierer liefert Qualitätsinformation für jede Base
  → Verbesserung der Mappingqualität durch Verwendung in Read Mapping (mit bisherigen Tools nicht möglich)
- Bisher kein Benchmark für vergleichende Analysen der Read Mapping Tools verfügbar



Data Management Challenges in NGS

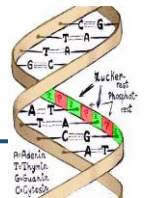
Literatur: Wandelt, Rheinländer, Bux, Thalheim, Haldemann, Leser: Data Management Challenges in Next Generation Sequencing. *Datenbank-Spektrum*, 2012.



1) Bestimmung des Read Alignments/Mappings
   - Viele kurze NGS ‚reads' müssen gegen Referenzsequenz (z.B. humanes Genom) gemappt werden → erfordert effiziente Lösungen!

2) Sequenzkompression
   - Speicherung und Übertragung von Genomdaten wird problematisch durch NGS
     - 1000 Genome → 3 Terabyte; noch wesentlich mehr mit Metadaten, Qualitätsinformationen etc.
     - Limitierungen der Bandbreite behindern den Datentransfer zwischen versch. Infrastrukturen, z.B. für übergreifende Analysen

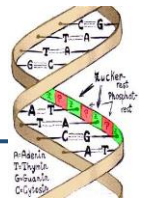
3) Scientific Workflows
   - Gleichzeitige Analyse von 100en Genomen
   - Komplexe Pipelines zahlreicher Tools



Sequenzkompression

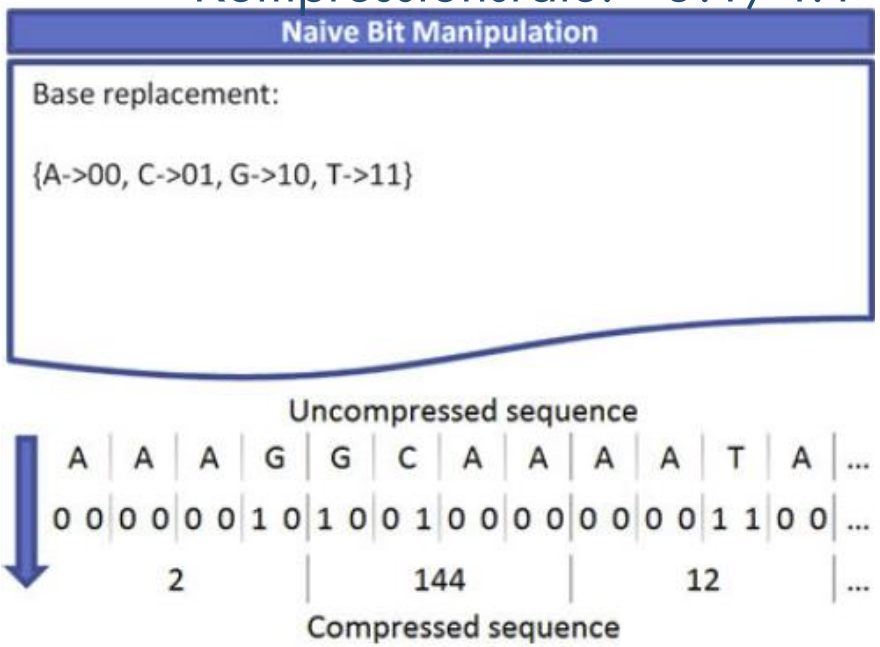
- Ziele
  - Ökonomische Speicherung
  - Effiziente Übertragung von Sequenzen
  - Einfacher Zugriff auf Sequenzen

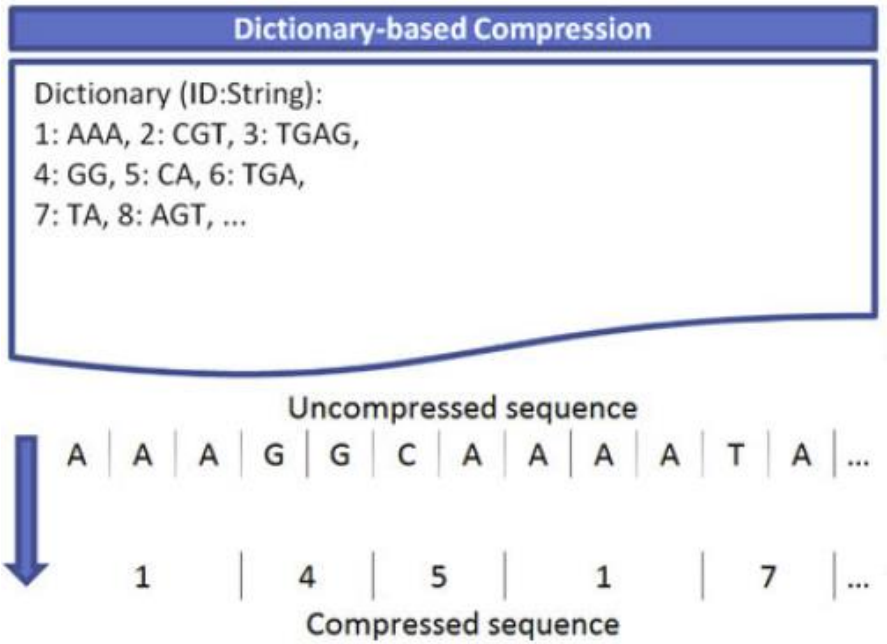
→ Verschlüsselung und Dekodierung biologischer Sequenzen



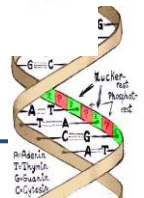
Traditionelle Kompressionsalgorithmen

- Beispiele
  - Naive Bit Manipulation: kodieren von >2 Zeichen in einem Byte
  - Dictionary-based Compression: Ersetzen langer, sich wiederholender Substrings durch Referenzen auf ein Wörterbuch (Aufbau zur Laufzeit)
  - Kompressionsrate: ~3:1/4:1





Wandelt, Rheinländer, Bux, Thalheim, Haldemann, Leser:
Data Management Challenges in Next Generation Sequencing. *Datenbank-Spektrum*, 2012.



Referenzielle Kompression

- Kodieren der Inputsequenz bzgl. einer externen Referenzsequenz (oder Menge von Sequenzen)
- Unterschied zu Dictionary-based: fixe Referenzsequenz, Referenzsequenz wird nicht in komprimierte Datei eingefügt
- Kompressionsrate: bis zu 400:1 (wenn Referenz evolutionär kurze Distanz zur komprimierten Sequenz hat)
- Vorherige Indexierung der Referenzsequenz
  → effizientes Finden langer Übereinstimmungen
- Tupel (Startposition, Subsequenzlänge)

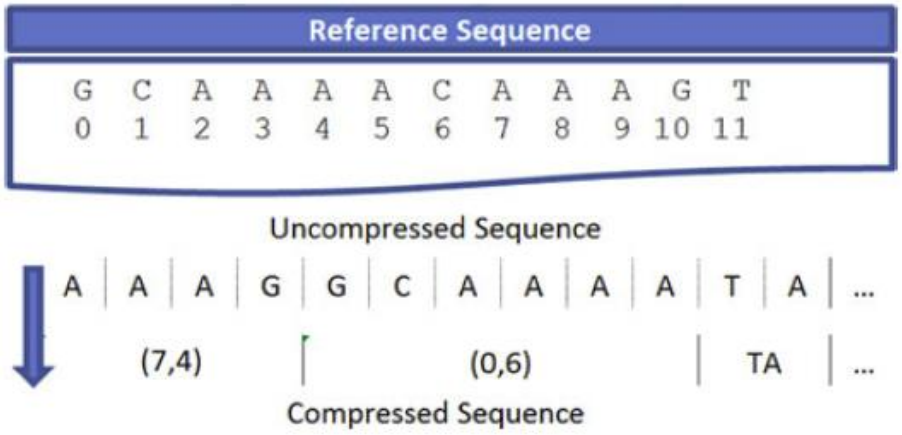
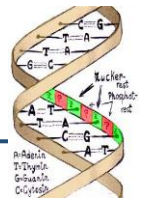
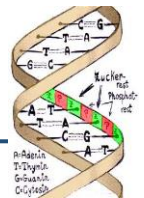


Abbildung: Wandelt, Rheinländer, Bux, Thalheim, Haldemann, Leser: Data Management Challenges in Next Generation Sequencing. *Datenbank-Spektrum*, 2012.



Referenzielle Kompression

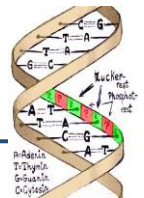
- Indexierung birgt Speicherproblem
  → Limitierung durch Festplattenzugriffe
- Suchen nach Übereinstimmung in lokaler Umgebung vorheriger Matches (biologisch begründet)
  → kein Zugriff auf Indexstruktur nötig (auch wenn Kompression durch längere Matches besser sein könnte)
- Splitten von Input und Referenz in Blöcke fester Größe
  - Jeden Block separat abhandeln
  - Weniger Hauptspeicher nötig
  - Parallelisierbare Kompression



Anforderungen Sequenzkompression

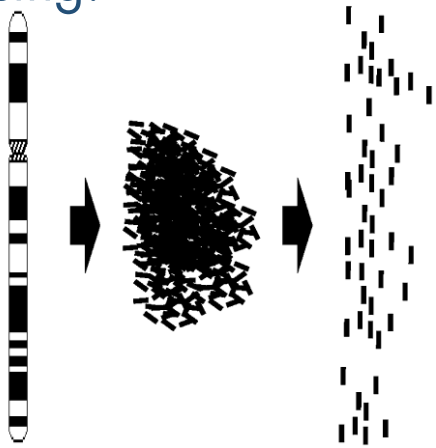
- Gute/hohe Skalierbarkeit, Kompressionsrate, Kompressionsgeschwindigkeit
- Finden der besten Referenzsequenz für 1000e Sequenzen
- Read Kompression: auch Quality Score abspeichern (Verlustbehaftete Kompression)
- Direkte Analyse komprimierter Sequenzen (ohne Dekomprimierung): nichts gewonnen, wenn 1000 komprimierte Sequenzen vor Analyse alle dekomprimiert werden müssen

  → brauchen String-Suchalgorithmen, die direkt die existierenden Indexstrukturen einer Referenzsequenz und die referentiell komprimierten Dateien nutzen können

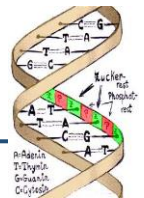


Data Management Challenges in NGS

Literatur: Wandelt, Rheinländer, Bux, Thalheim, Haldemann, Leser: Data Management Challenges in Next Generation Sequencing. *Datenbank-Spektrum*, 2012.



1) Bestimmung des Read Alignments/Mappings
   - Viele kurze NGS ‚reads' müssen gegen Referenzsequenz (z.B. humanes Genom) gemappt werden → erfordert effiziente Lösungen!
2) Sequenzkompression
   - Speicherung und Übertragung von Genomdaten wird problematisch durch NGS
     - 1000 Genome → 3 Terabyte; noch wesentlich mehr mit Metadaten, Qualitätsinformationen etc.
     - Limitierungen der Bandbreite behindern den Datentransfer zwischen versch. Infrastrukturen, z.B. für übergreifende Analysen
3) Scientific Workflows
   - Gleichzeitige Analyse von 100en Genomen
   - Komplexe Pipelines zahlreicher Tools



Scientific Workflows

- Scientific Workflow Management Systems, z.B. Taverna, Galaxy

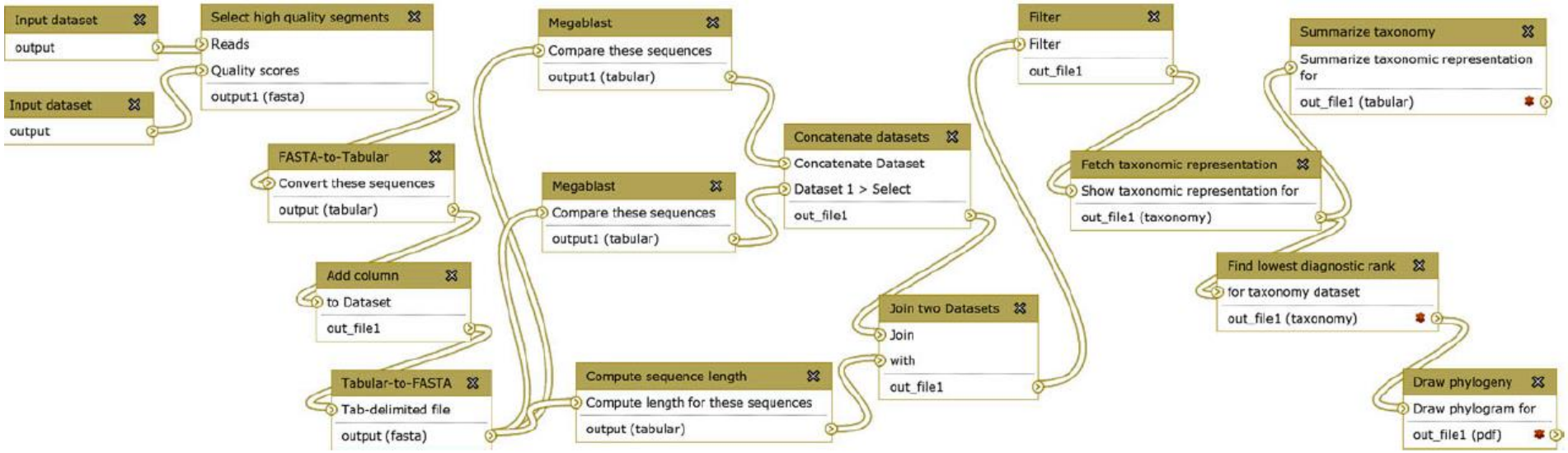


Abbildung: Wandelt, Rheinländer, Bux, Thalheim, Haldemann, Leser:
Data Management Challenges in Next Generation Sequencing. *Datenbank-Spektrum*, 2012.

NGS Analyse Pipelines

- Meist der erste Schritt: Rekonstruktion der Sequenz aus reads: Assemblierung oder Read Alignment (gegen Referenzgenom)
- Bisher keine Standards, sehr viele Tools (z.B. für Alignment) verfügbar
- Verwenden versch. Tools in Kombination, zur Verbesserung der Qualität
- Weitere Schritte abhängig von der Fragestellung, z.B. Bestimmung von Varianten (z.B. SNV=single nucleotid variants) → Finden einer Mutation, die für Krankheit verantwortlich sein könnte (Assoziationen)

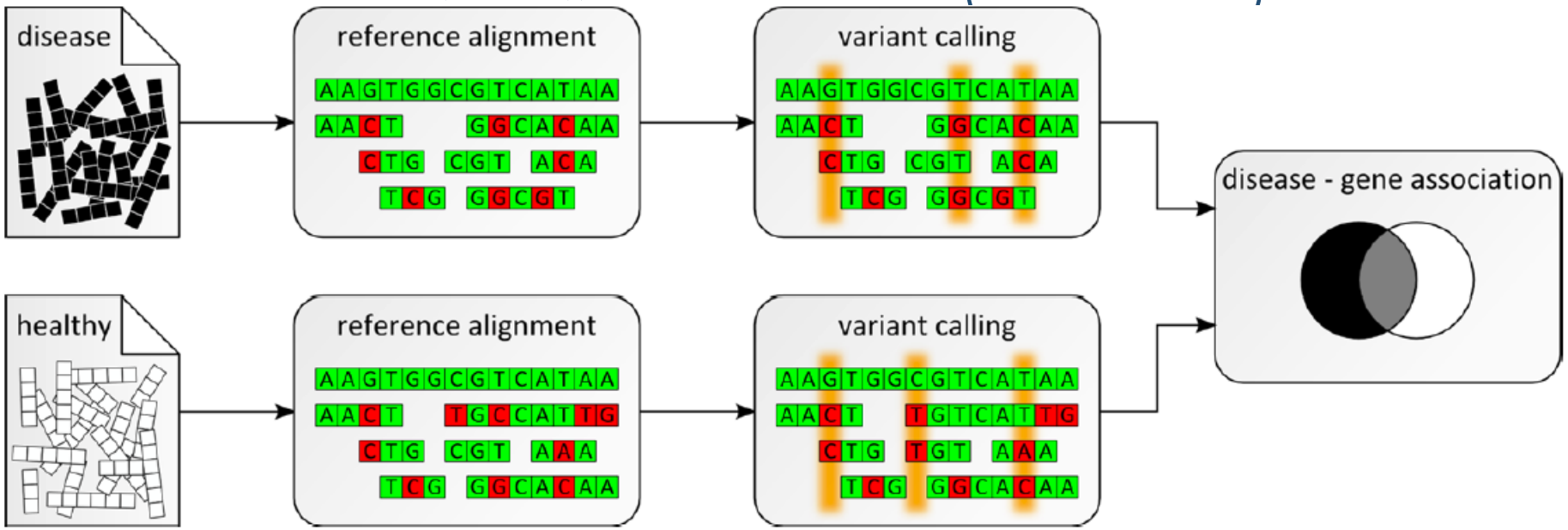
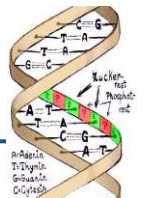


Abbildung: Wandelt, Rheinländer, Bux, Thalheim, Haldemann, Leser: Data Management Challenges in Next Generation Sequencing. *Datenbank-Spektrum*, 2012.

Parallelisierung von Scientific Workflows

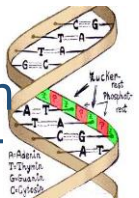
- Taskparallelität: verschiedene Tasks auf parallelen (unabhängigen) „Zweigen“ werden auf verschiedene, unabhängige Computing Nodes verteilt
- Pipelining: sequenzielle Datenprozessierungstasks werden simultan auf verschiedenen Fragmenten der Inputdaten ausgeführt (Datenfluss durch Workflow, ähnlich Öl-Pipeline)
- Datenparallelität: gesamte Inputdaten werden partitioniert + der ganze Workflow wird für jedes Datenfragment repliziert
- Die teuren Operationen (Assembly, read Alignment, Variant Calling) lassen sich gut parallelisieren
  - Workflow Management Systeme (Taverna): nur Pipelining (keine Task-/Datenparallelität)
  - Cluster/Grid scheduler (Pegasus): Taskparallelität (meist kein Pipelining, Datenparallelität)
  - Dataflow Engine (PACT/Nephele, PIG/HADOOP): Datenparallelität (restriktiv: alle Pipelines sind Folge von map/reduce-tasks)

  → Kein System, das Pipelining, Task-und Datenparallelität unterstützt



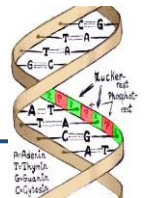
Offene Probleme

- Cloud Computing für Big Data Analysen
  - Transfer der Input & Output Daten in die/aus der Cloud
    - Kompression
    - Referenzgenome direkt in Cloud verfügbar machen
      z.B. Genbank auf EC2 (nicht hilfreich für neue Sequenzen)
  - Nicht zufriedenstellend gelöst: Effizientes Mappen der Workflow Tasks auf heterogene, verteilte Knoten (z.B. virtuelle Maschinen in der Cloud)
    - Datentransferzeit muss einbezogen werden!
    - Workflow Scheduler muss Ausführen eines Workflow in dynamischer Umgebung ständig abstimmen/steuern (Bandbreite, verfügbarer Speicher, Geschwindigkeit ändern sich ständig)
    - Cloud „richtig nutzen“ (Elastizität):
      automatische Entscheidung während Workflowausführung, möglichst viele Ressourcen für aufwendige Aufgaben zu nutzen



Offene Probleme

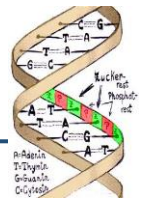
- Management von Metadaten
- Datenintegration (z.B. Integration genomischer Daten mit Funktionen und Interaktionen)
- Datenschutz: Anonymisierung der Sequenzdaten!
  - Komplex, da Person über Sequenz eindeutig identifiziert werden kann (→ nur Name löschen/ersetzen reicht nicht);
  - non-public „walled“ cloud solutions mit strikter Zugriffskontrolle



Zusammenfassung

- Methoden des Next Generation Sequencing erlauben parallele Sequenzierung von Millionen - Milliarden von reads
  - Produzieren sehr große Datenmengen
  - Neue Möglichkeiten für personalisierte und translationale Medizin (z.B. Diagnose mit NGS), zielgerichtete Untersuchung von Mutationen und Polymorphismen, …

- Herausforderungen für Datenmanagement:
  - Bestimmung des Read Mappings
    - Effiziente Bestimmung mittels Parallelisierung und Indexierung
  - Sequenzkompression
    - Effiziente Übertragung und Speicherung von Genomdaten
  - Scientific Workflows
    - Komplexe Pipelines zahlreicher Tools
    - Gleichzeitige Analyse hunderter Genome



Fragen ?