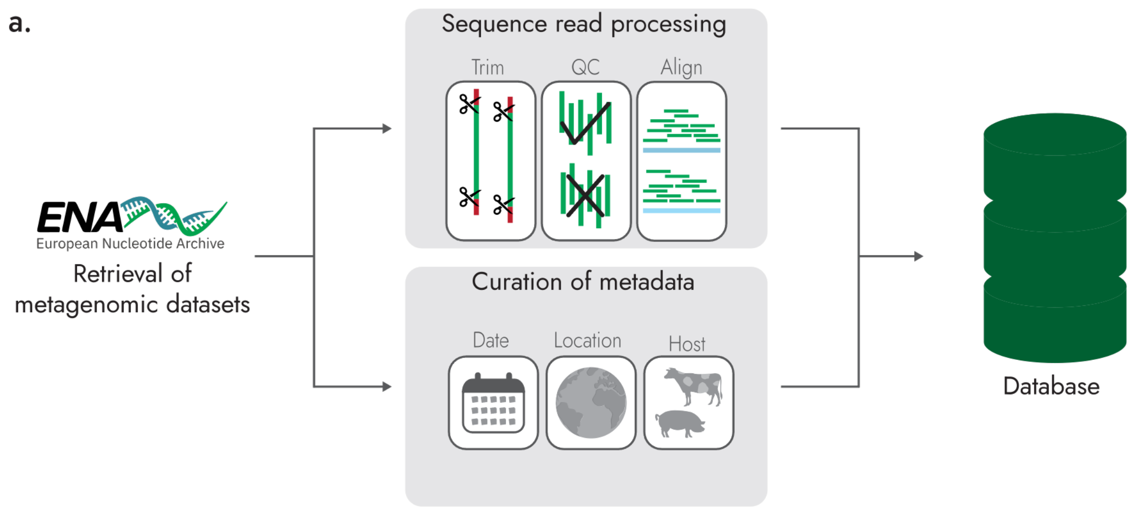
```
pos[0] = 0;
for(m = t_len - 1, nuc_pos = t_e - 1; m >= 0; --m, --nuc_pos) {

        if(nuc_pos < 0) {
                nuc_pos = template_length - 1;
        }

        D_ptr[q_len] = (0 < k) ? 0 : (W1 + (t_len - 1 - m) * U);
        Q_prev = (t_len + q_len) * (MM + U + W1);

        t_nuc = getNuc(template, nuc_pos);
        for(n = q_len - 1; n >= 0; --n) {
                E_ptr[n] = 0;

                /* update Q and P, gap openings */
                Q = D_ptr[n + 1] + W1;
                P_ptr[n] = D_prev[n] + W1;
                if(Q < P_ptr[n]) {
                        D_ptr[n] = P_ptr[n];
                        e = 4;
                } else {
                        D_ptr[n] = Q;
                        e = 2;
                }

                /* update Q and P, gap extensions */
                /* mark bit 4 and 5 as possible gap-opennings, if necessarry */
                thisScore = Q_prev + U;
                if(Q < thisScore) {
                        Q = thisScore;
                        if(e == 2) {
                                D_ptr[n] = Q;
                                e = 3;
                        }
                } else {
                        E_ptr[n] |= 16;
                }
                thisScore = P_prev[n] + U;
                if(P_ptr[n] < thisScore) {
                        P_ptr[n] = thisScore;
                        if(D_ptr[n] < thisScore) {
                                D_ptr[n] = thisScore;
                                e = 5;
                        }
                } else {
                        E_ptr[n] |= 32;
                }

                /* Update D, match */
                thisScore = D_prev[n + 1] + d[t_nuc][query[n]];
                if(D_ptr[n] < thisScore) {
                        D_ptr[n] = thisScore;
                        E_ptr[n] |= 1;
                } else {
                        E_ptr[n] |= e;
                }

                Q_prev = Q;
        }

        E_ptr -= (q_len + 1);

        if(k < 0 && Stat.score <= *D_ptr) {
                Stat.score = *D_ptr;
                pos[0] = m;
        }

        tmp = D_ptr;
        D_ptr = D_prev;
        D_prev = tmp;

        tmp = P_ptr;
        P_ptr = P_prev;
        P_prev = tmp;
}
E_ptr = E;
```

Targeted Metagenomic Assemblies

Philip T.L.C. Clausen

Stats from AvA 1.0

- 214,095 metagenomic sequencing runs mapped

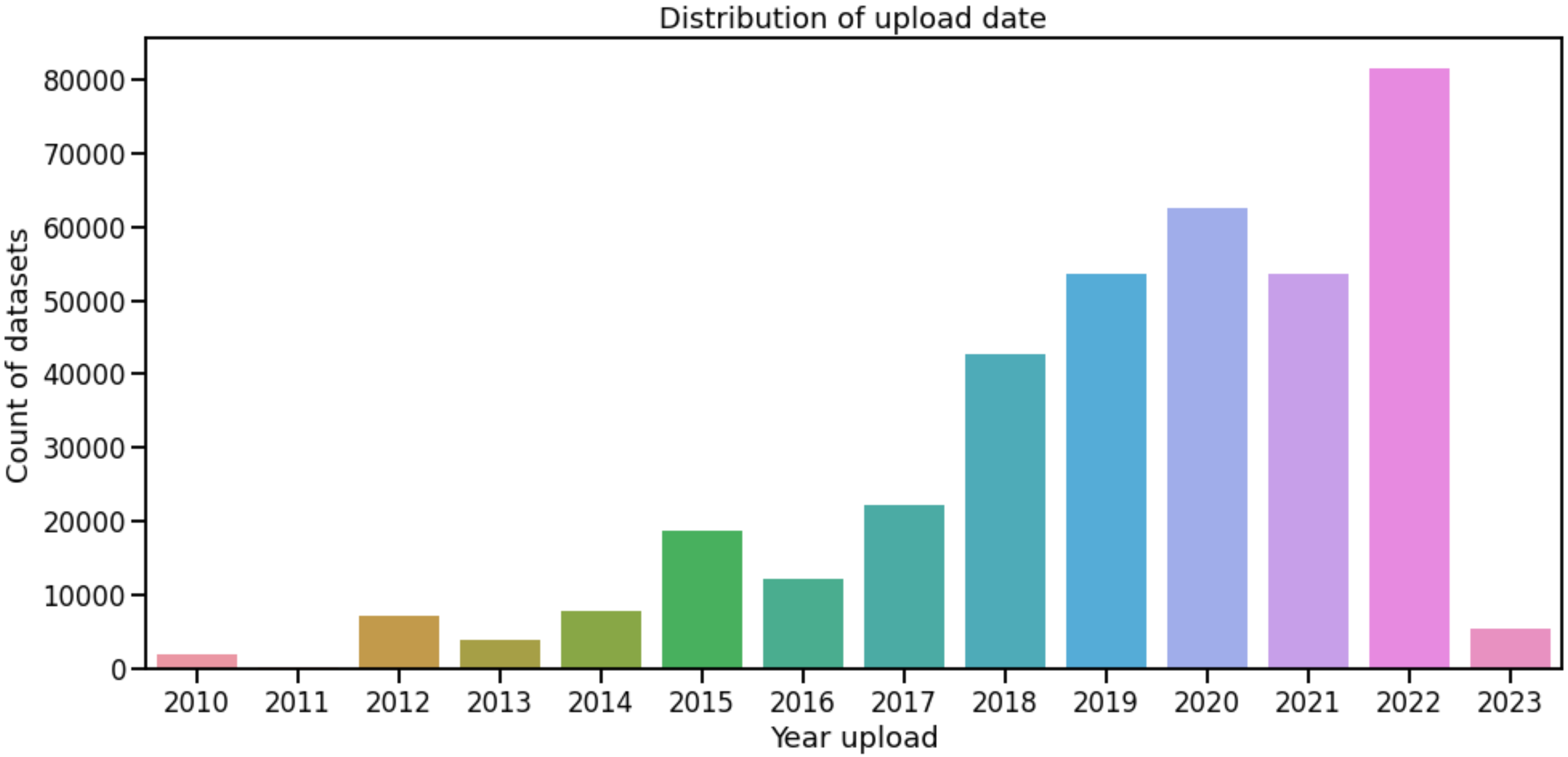


- Of these, 869 *mcr* metagenomes were assembled with SPAdes

But this time
Bigger and Better

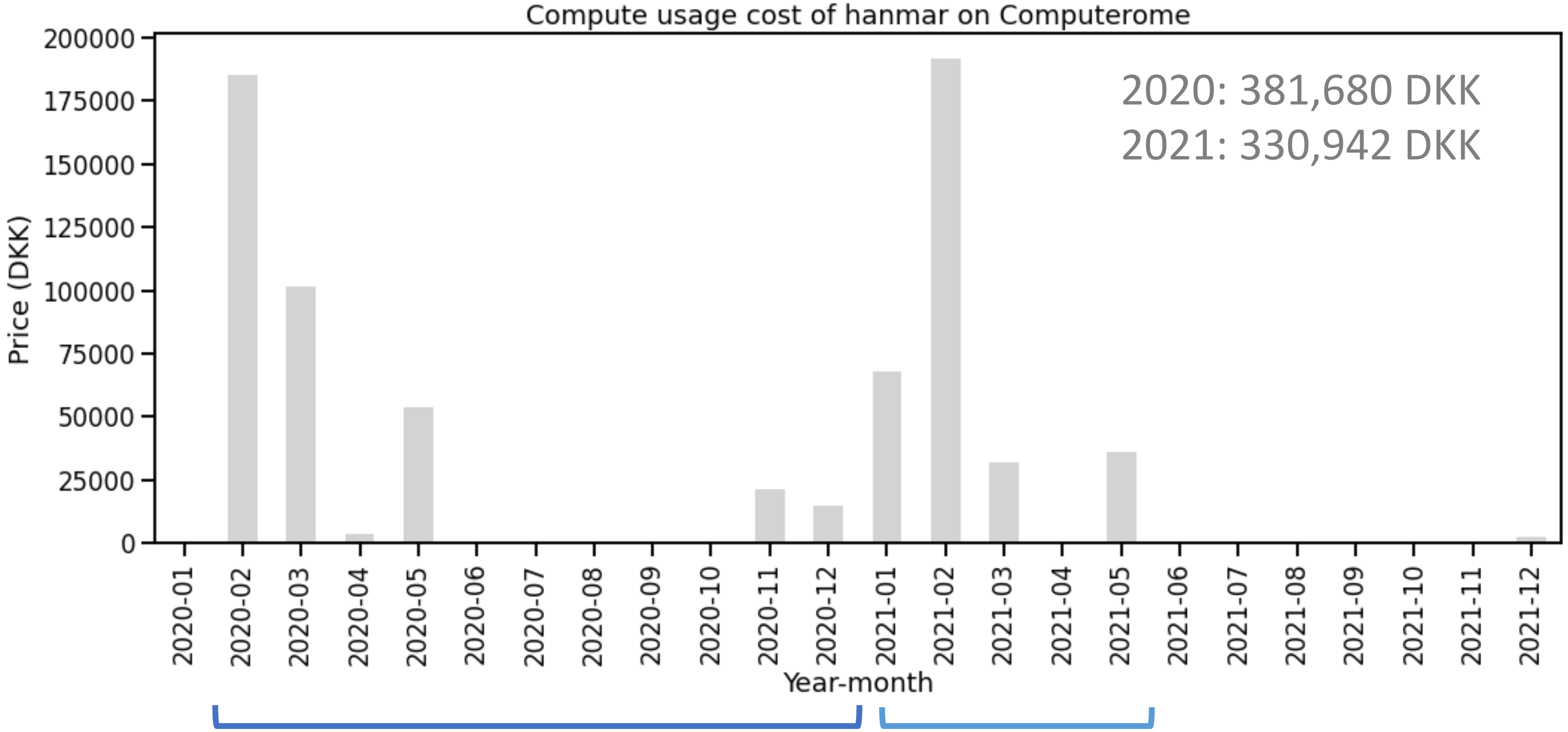
Metagenomic data from ENA

The number of datasets has continued to grow… **356,132** (772 TB) per 20/01/2023



Stats from AvA 1.0

- 214,095 metagenomic sequencing runs



2020: 381,680 DKK
2021: 330,942 DKK

Bulk of mapping
(~131k runs)

869 De-novo assemblies
of *mcr* metagenomes

Project ideas

🟦 assembly ▨ abundance ⬛ other

Disproving "everything is everywhere but the environment selects"
🟦▨ Frank

Identifying hotspots or putative origins of ARGs
🟦 Thomas, Markus

Identification of the source and sink locations for the global emergence and spread of resistance determinants/ pathogen variants of interest
🟦 Patrick N

Characterizing abundance of ARGs to novel antibiotics
🟦▨ Hannah (Csaba Pal)

Susceptibility-resistance co-occurrences
🟦 Amalia

AMR genes and host-jump
🟦 Shinny

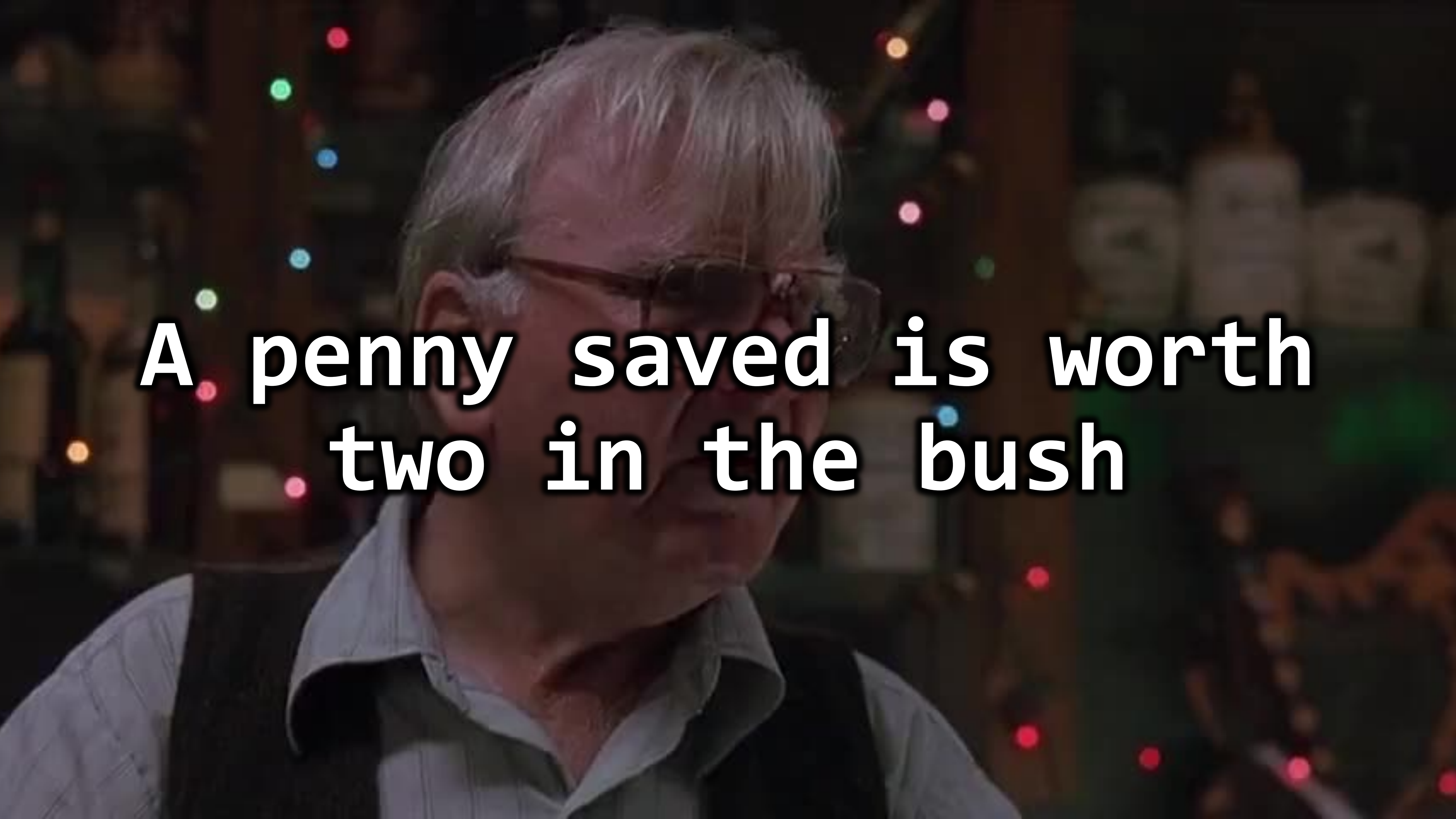
Macro-ecology of microbiomes
Hannah, Saria (David Bravo Nogues)
▨

Statistical determination of mutation and selection of genes across environments (Million Miles High Alignment)
Patrick M
▨

Abundances in the global resistome
▨ Hannah

Long-term storage structures
⬛ Timmie, Malte

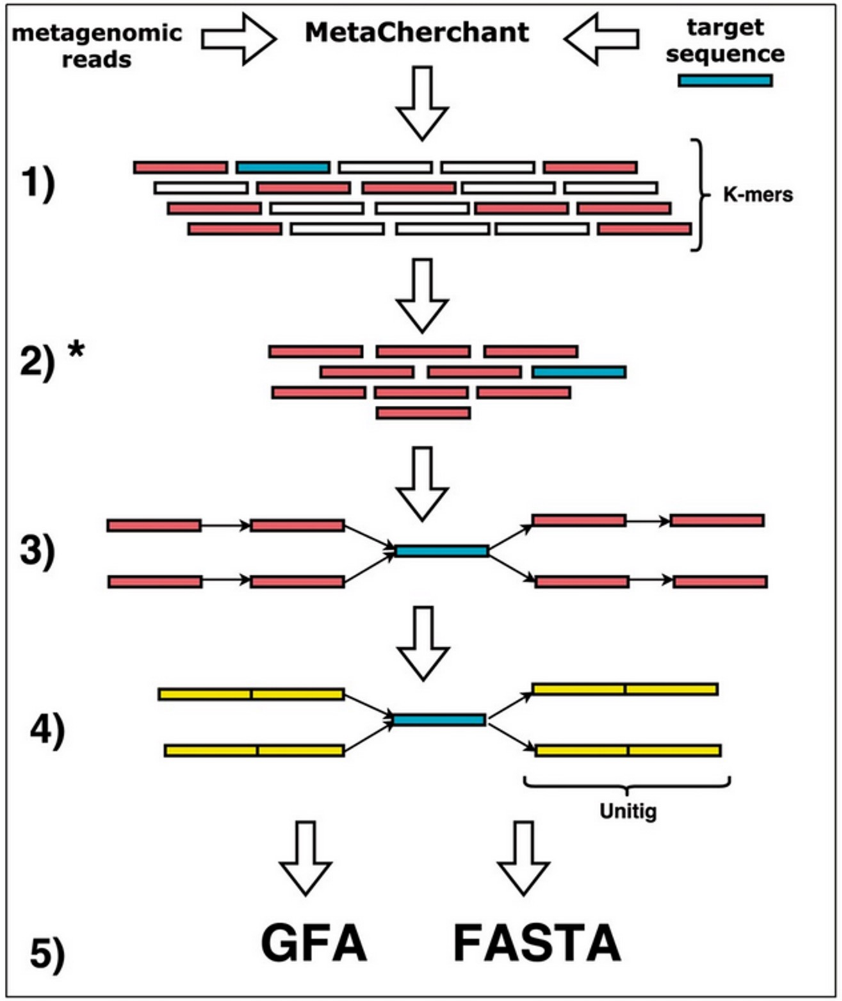
https://docs.google.com/document/d/1GjTxGFAJl_dn9NIs7h2CFjcmaeJR2oc5QE--Rgx_zpU/edit?usp=sharing



A penny saved is worth two in the bush

Metacherchant

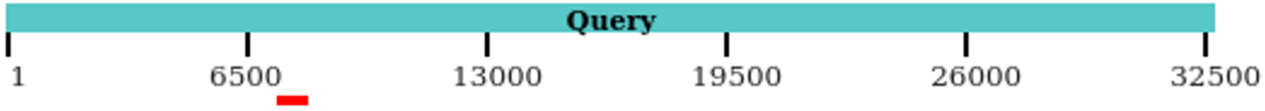
- Algorithm for extracting the genomic environment of antibiotic resistance genes
- Performs sensitive taxonomic classification of sequencing reads from metagenomes



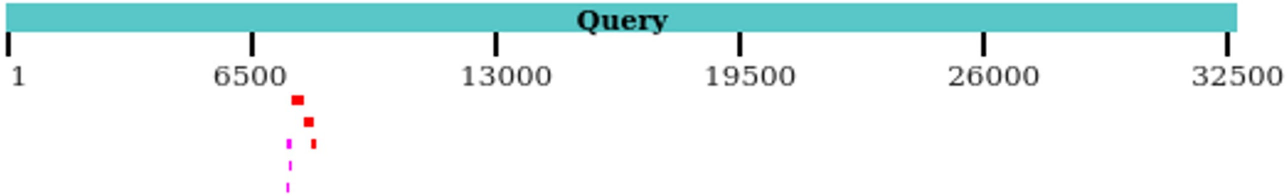
Validation of Metacherchant results

Fast (~5 min) and somewhat efficient (~30 GB peak memory)

Spades node containing *blaTEM vs blaTEM* gene



Spades node containing *blaTEM vs* Metacherchant reported results



Don’t cross the road if you can’t get out of kitchen

Targeted Metagenomic Assembly

- Target sequence(s) of concern
- Metagenomic sample / raw reads
- Assembly around the target sequence(s)

What do we have to do that

- KMA can identify the target sequence(s) to satisfaction with reasonable resource consumption.

- SPAdes can perform metagenomic assemblies with satisfiable quality but using computational resources we do not have.

SPAdes

- Builds de Bruijn graphs, which is used to reconstruct larger pieces of genomic content.
- Scales with the number of unique *k*-mers and number of reads.
- **We just need to reduce the number of reads.**

First shot

1. Align reads to target(s) using KMA
2. If no increase in aligning reads continue from 7.
3. Extract aligning read(s)
4. De novo assemble read(s) using SPAdes
5. Set de novo assembly as new target
6. Redo from 1.
7. Save last assembly as final targeted assembly.

First problem(s)

C to the rescue: fqgrep

1. Align reads to target(s) using KMA
2. If no increase in aligning reads continue from 7.
3. Extract aligning read(s)
4. De novo assemble read(s) using SPAdes
5. Set de novo assembly as new target
6. Redo from 1.
7. Save last assembly as final targeted assembly.

First problem(s)

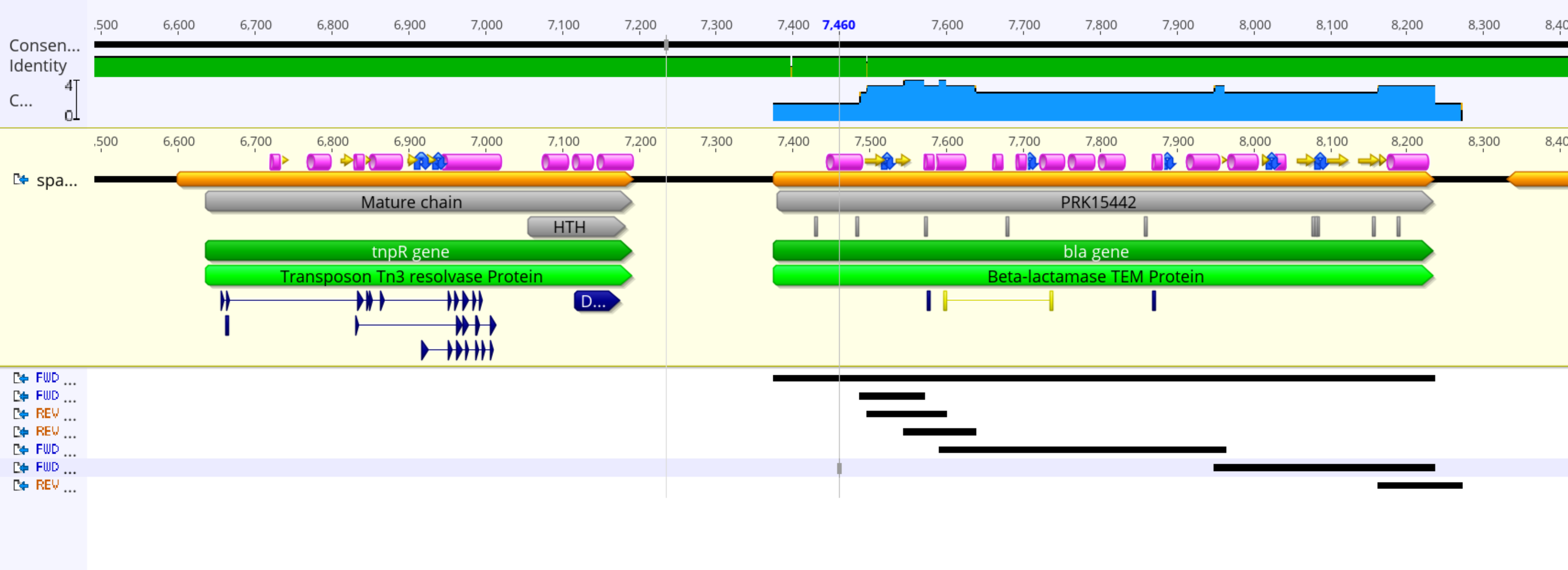
Set new target as scaffolds aligning to original target.

1. Align reads to target(s) using KMA
2. If no increase in aligning reads continue from 7.
3. Extract aligning read(s)
4. De novo assemble read(s) using SPAdes
5. Set de novo assembly as new target
6. Redo from 1.
7. Save last assembly as final targeted assembly.

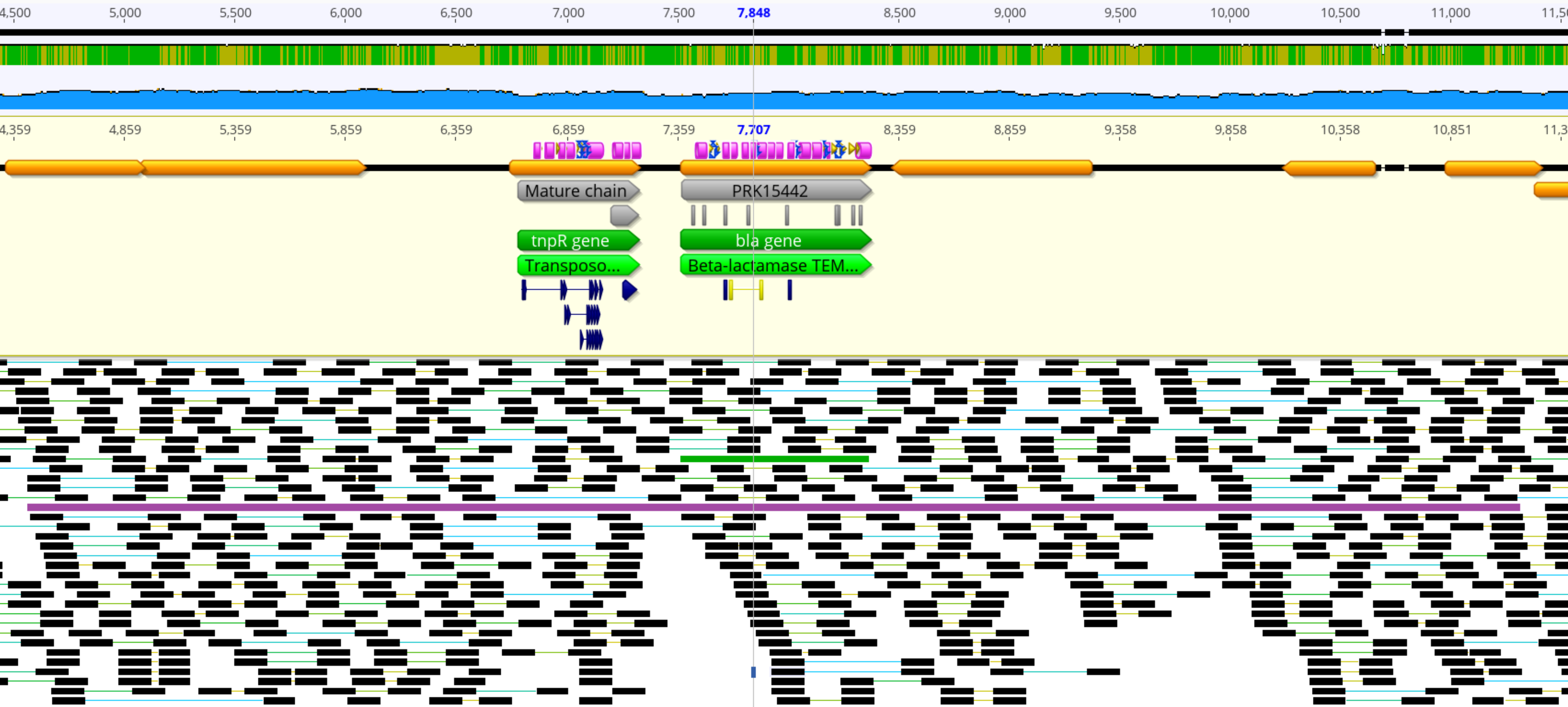
A better shot

1. Align reads to target(s) using KMA
2. If no increase in aligning reads continue from 7.
3. Extract aligning read(s) using fqgrep
4. De novo assemble read(s) using SPAdes
5. Set scaffolds aligning to org. target as new target
6. Redo from 1.
7. Save last assembly as final targeted assembly.

But does it work?



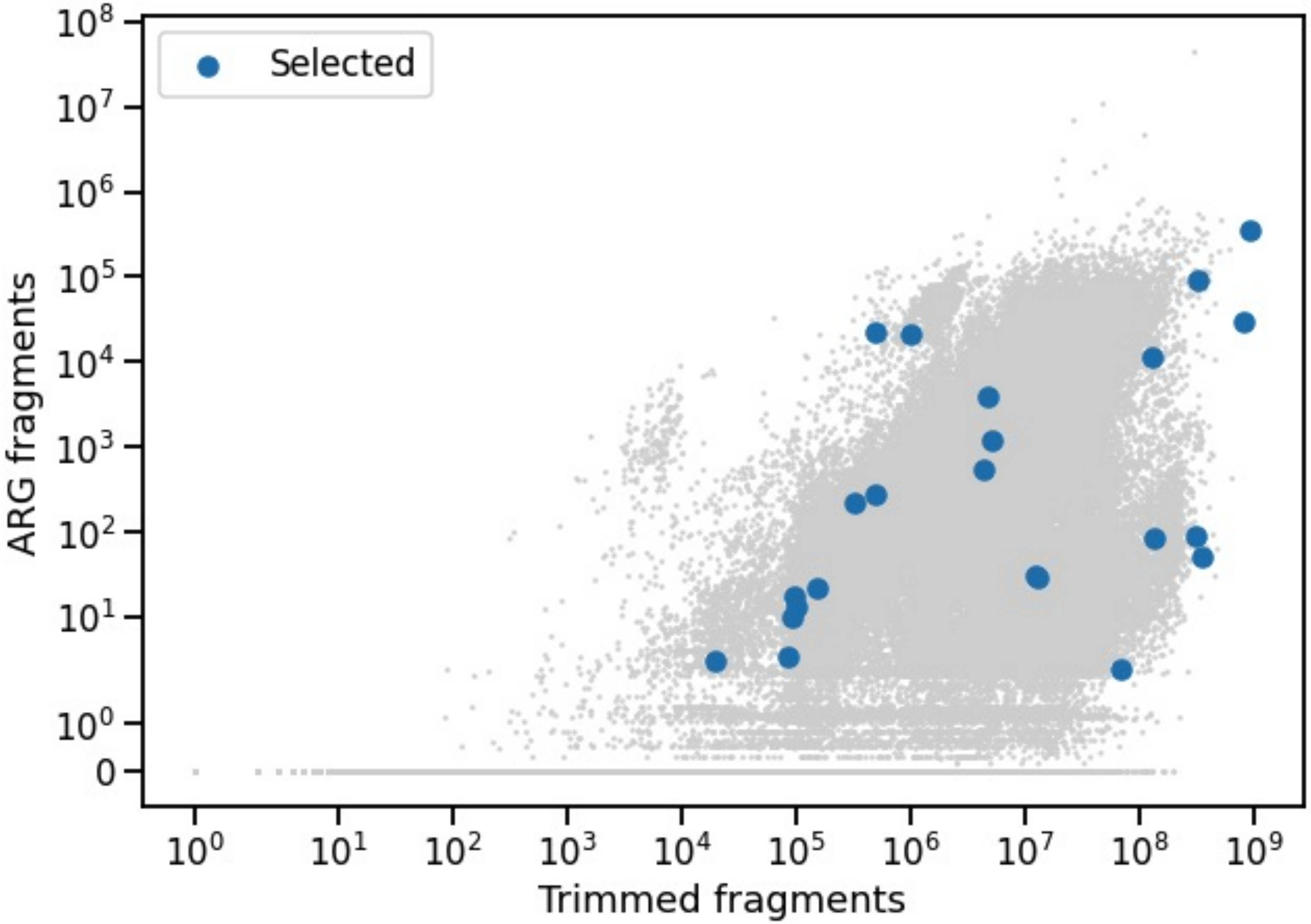
But does it work?



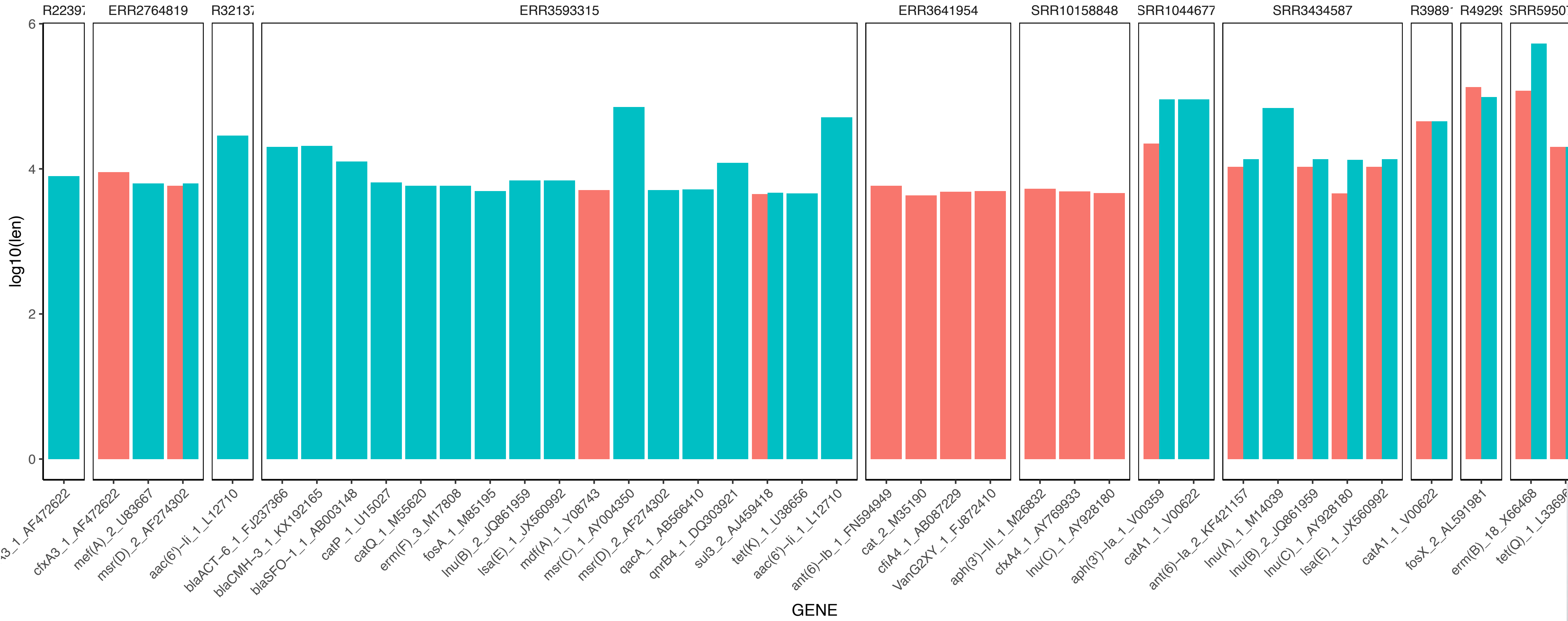
How about resources?

- < 700 MB
- Usually around 15 minutes

Some more test’s



Contigs with at least 1.5 kbp flanks

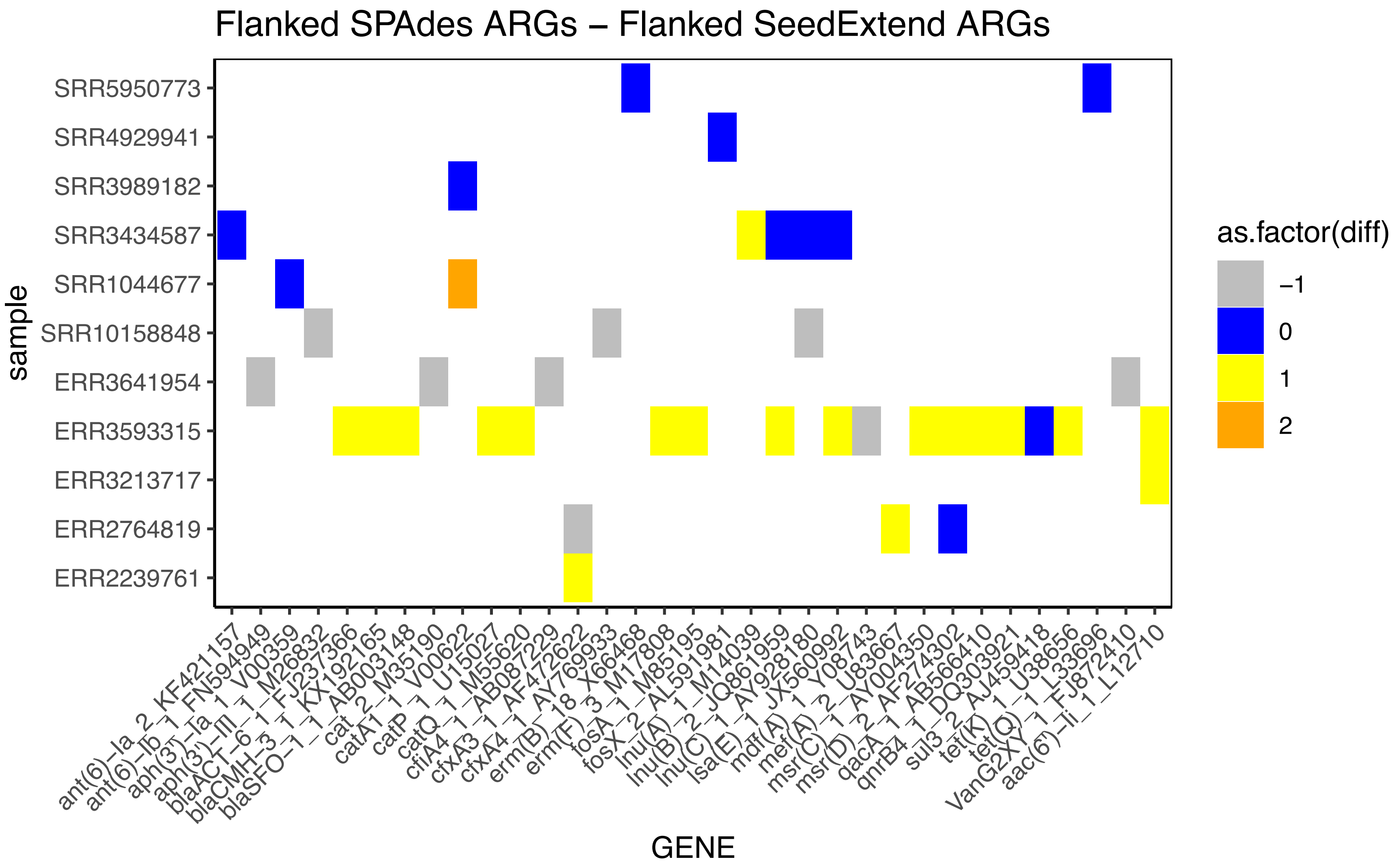


method

🟥 extend

🟦 spades

Contigs with at least 1.5 kbp flanks



Bottleneck(s)

1. Align reads to target(s) using KMA
2. If no increase in aligning reads continue from 7.
3. Extract aligning read(s) using fqgrep
4. De novo assemble read(s) using SPAdes
5. Set scaffolds aligning to org. target as new target
6. Redo from 1.
7. Save last assembly as final targeted assembly.

Bottleneck on large samples with few hits:

unzipping

1. Align reads to target(s) using KMA
2. If no increase in aligning reads continue from 7.
3. Extract aligning read(s) using fqgrep
4. De novo assemble read(s) using SPAdes
5. Set scaffolds aligning to org. target as new target
6. Redo from 1.
7. Save last assembly as final targeted assembly.

Bottleneck on samples with many hits:

1. Align reads to target(s) using KMA
2. If no increase in aligning reads continue from 7.
3. Extract aligning read(s) using fqgrep
4. De novo assemble read(s) using SPAdes
5. Set scaffolds aligning to org. target as new target
6. Redo from 1.
7. Save last assembly as final targeted assembly.

What we get.

- Assembly with targeted gene(s)
- Assembly graph
- Tsv with information about which targets are located on which contigs.

People in glass houses
SINK SHIPS