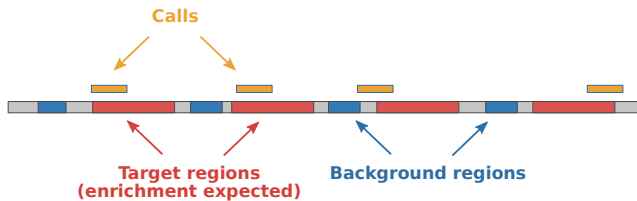


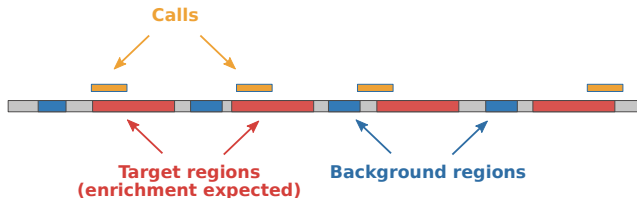
**The algorithm of perm-test**  
<https://github.com/pd3/utls/>

# Enrichment test



Are calls enriched in the target regions?

# Enrichment test

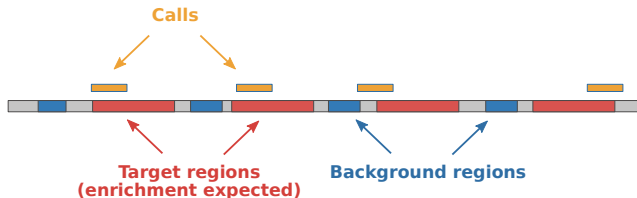


Are calls enriched in the target regions?

Standard permutation test

- randomly place the calls on the genome and count how many are placed in the target regions. Repeat many times and count how often the number of calls placed in the target regions matched or exceeded the count observed in the experimental data

# Enrichment test



Are calls enriched in the target regions?

Standard permutation test

- ▶ randomly place the calls on the genome and count how many are placed in the target regions. Repeat many times and count how often the number of calls placed in the target regions matched or exceeded the count observed in the experimental data



Computational problem

- ▶ when working with exomes (2% of the genome), calls are placed in inaccessible regions 98% of the time. A huge waste of CPU time!

## Enrichment test: naive improvement

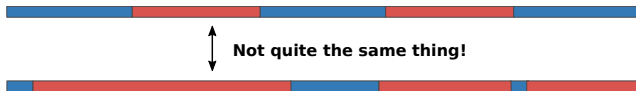


We could just permute labels, that's very fast!

## Enrichment test: naive improvement

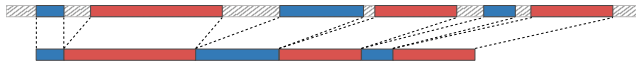


We could just permute labels, that's very fast!



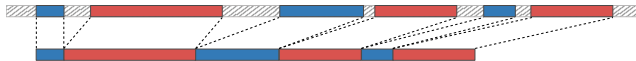
However, if we only permute the labels, we ignore size differences between target and background regions.

## Trimmed genome

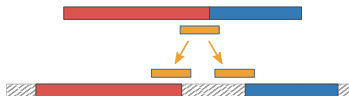


Instead, we splice out inaccessible regions creating a shorter version of the genome. This way every random placement results in a usable call, it can never land in an inaccessible region.

## Trimmed genome



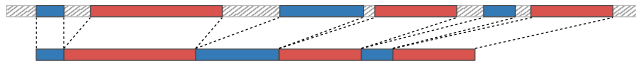
Instead, we splice out inaccessible regions creating a shorter version of the genome. This way every random placement results in a usable call, it can never land in an inaccessible region.



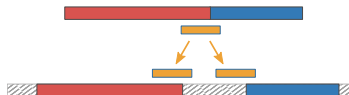
However, this leads to underrepresentation of region edges.



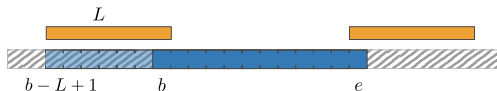
## Trimmed genome



Instead, we splice out inaccessible regions creating a shorter version of the genome. This way every random placement results in a usable call, it can never land in an inaccessible region.

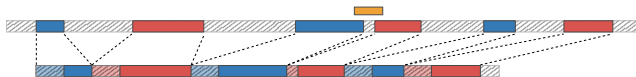


However, this leads to underrepresentation of region edges.



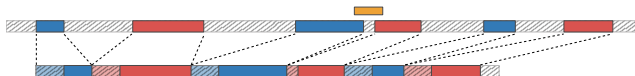
The trimmed genome must include left overhangs.

# Implementation



Be careful with calls that overlap more than one region in the real genome. Also take care at chromosome boundaries. Note that we must build a separate trimmed genome for each call length.

# Implementation



Be careful with calls that overlap more than one region in the real genome. Also take care at chromosome boundaries. Note that we must build a separate trimmed genome for each call length.

It would require too much memory to keep the genomes for all call lengths at the same time. Instead of updating all calls within a single iteration, iterate calls individually, one at a time, and keep in memory the counters of hits for all iterations.

```
for (iter=0; iter<niters; iter++)
{
  for (call=0; call<ncalls; call++)
    place_call_randomly

  if ( number_of_hits_exceeded )
    increase_pvalue
}
```

**Iterate all calls jointly  
update P-value in every iteration  
(the standard way)**

```
for (call=0; call<ncalls; call++)
{
  for (iter=0; iter<niter; iter++)
  {
    place_call_randomly
    if ( is_a_hit )
      increase_number_of_hits_in_the_iter
  }
}
for (iter=0; iter<niters; iter++)
if ( number_of_hits_exceeded )
  increase_pvalue
```

**Iterate calls individually,  
update P-value at the end  
(perm-test)**

# Usage

```
perm-test
  --calls          calls.txt
  --target-regs    tgt.txt
  --background-regs bg.txt
  --ref-fai        ref.fai    # required for chr lengths
  --niter          1e9,1e8    # iterations total and per batch
```

# Usage

```
perm-test
  --calls          calls.txt
  --target-regs    tgt.txt
  --background-regs bg.txt
  --ref-fai        ref.fai    # required for chr lengths
  --niter          1e9,1e8    # iterations total and per batch
```

The regions and calls should be a tab-delimited files with inclusive 1-based coordinates

```
1  69090  70008
1  861321 861496
1  865532 865787
etc
```

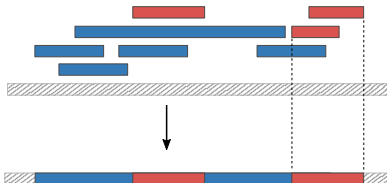
# Usage

```
perm-test
--calls          calls.txt
--target-regs    tgt.txt
--background-regs bg.txt
--ref-fai        ref.fai    # required for chr lengths
--niter          1e9,1e8    # iterations total and per batch
```

The regions and calls should be a tab-delimited files with inclusive 1-based coordinates

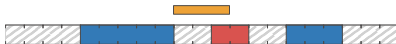
```
1  69090  70008
1  861321 861496
1  865532 865787
etc
```

The regions will be correctly merged and spliced when necessary



# Is it trustworthy?

We used many toy cases to calculate the probability exactly



# Is it trustworthy?

We used many toy cases to calculate the probability exactly

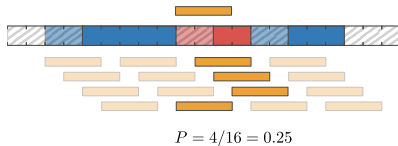


$$P = 4/16 = 0.25$$

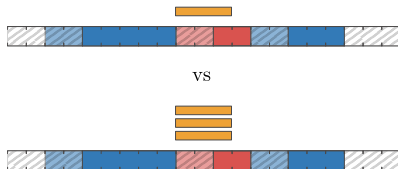


# Is it trustworthy?

We used many toy cases to calculate the probability exactly

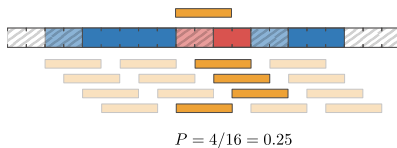


For complex cases we used identities such as  $P(c_1, \dots, c_n) = \prod P(c_i)$

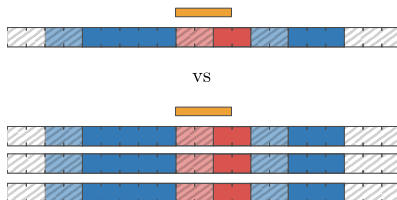


# Is it trustworthy?

We used many toy cases to calculate the probability exactly

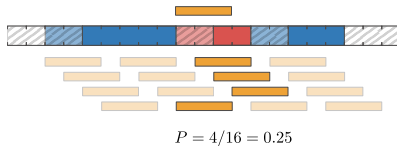


For complex cases we used identities such as  $P(c_1, \dots, c_n) = \prod P(c_i)$

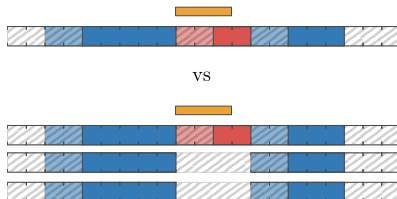


# Is it trustworthy?

We used many toy cases to calculate the probability exactly



For complex cases we used identities such as  $P(c_1, \dots, c_n) = \prod P(c_i)$



# Performance

- ▶ 4k iterations per call per minute
- ▶ 255 calls,  $10^{11}$  iterations, 140 hosts . . . 400 MB RAM, 5 hours
- ▶ flexible memory usage (can be lowered by decreasing the number of iterations per batch)