Public-key cryptography

Daniel J. Bernstein
Tanja Lange

Part I
Elliptic-curve crypto

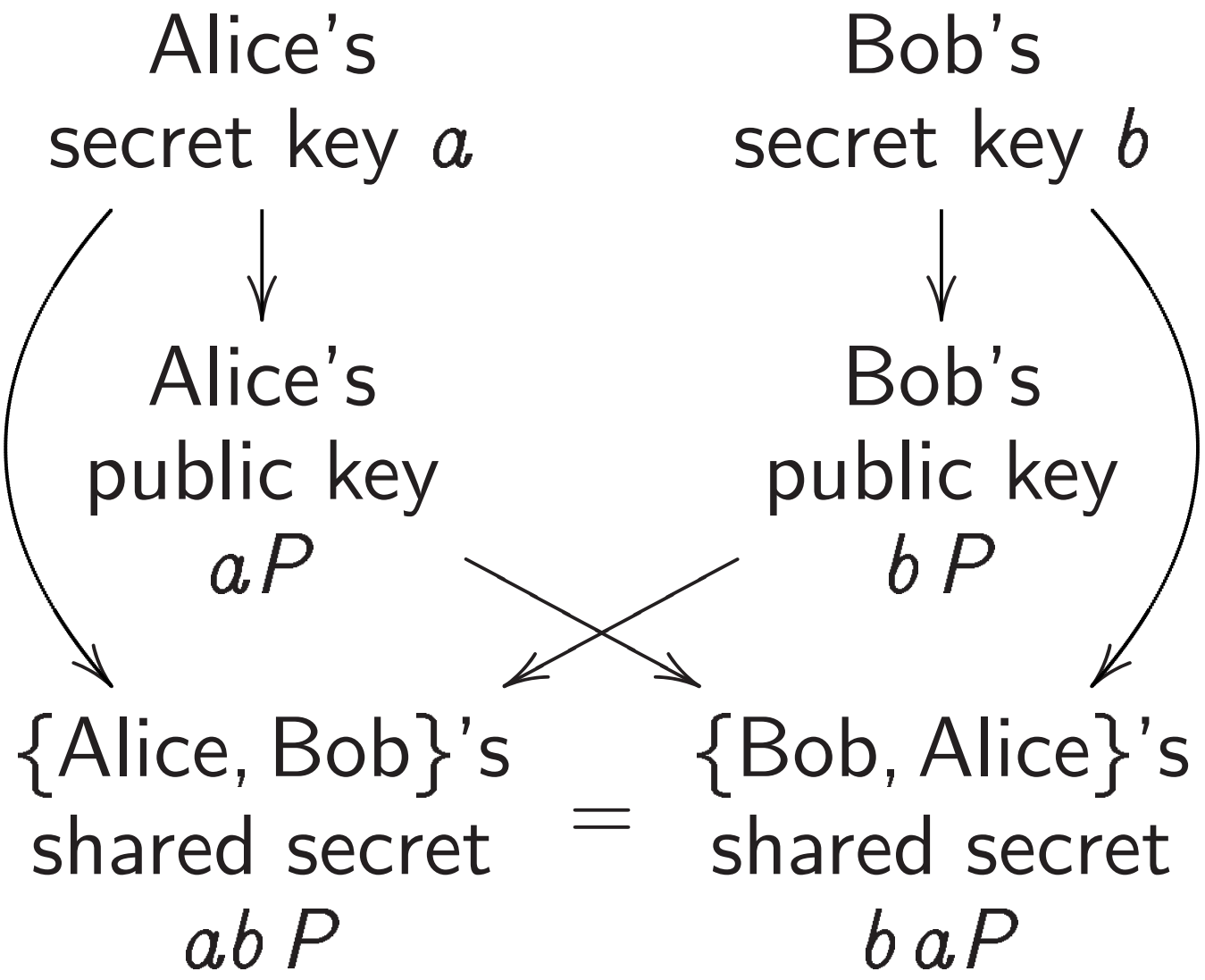
15 Aug 2017

Diffie-Hellman key exchange

Pick some *generator* P,

i.e. some group element

(using additive notation here).

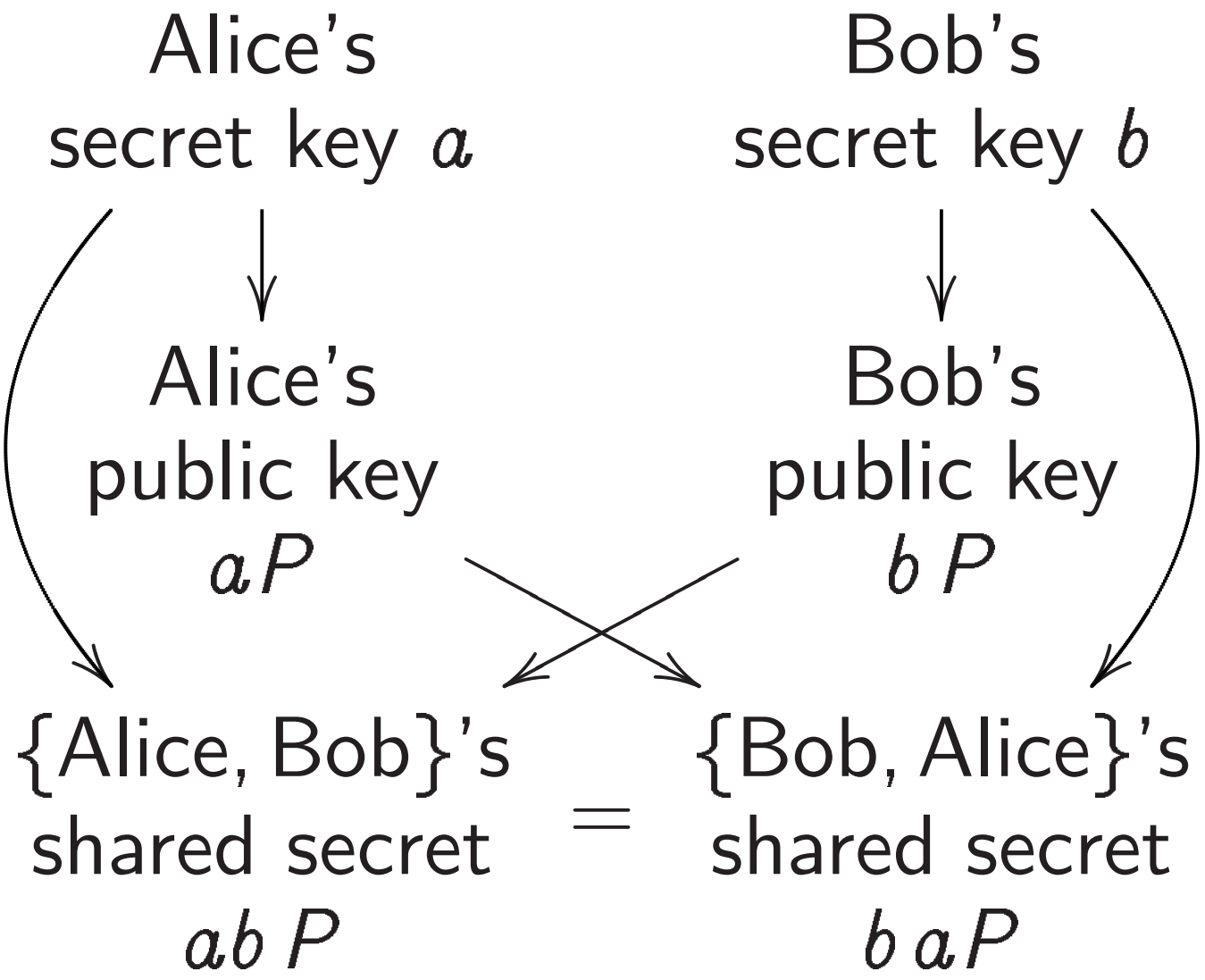


Diffie-Hellman key exchange

Pick some *generator* P,

i.e. some group element

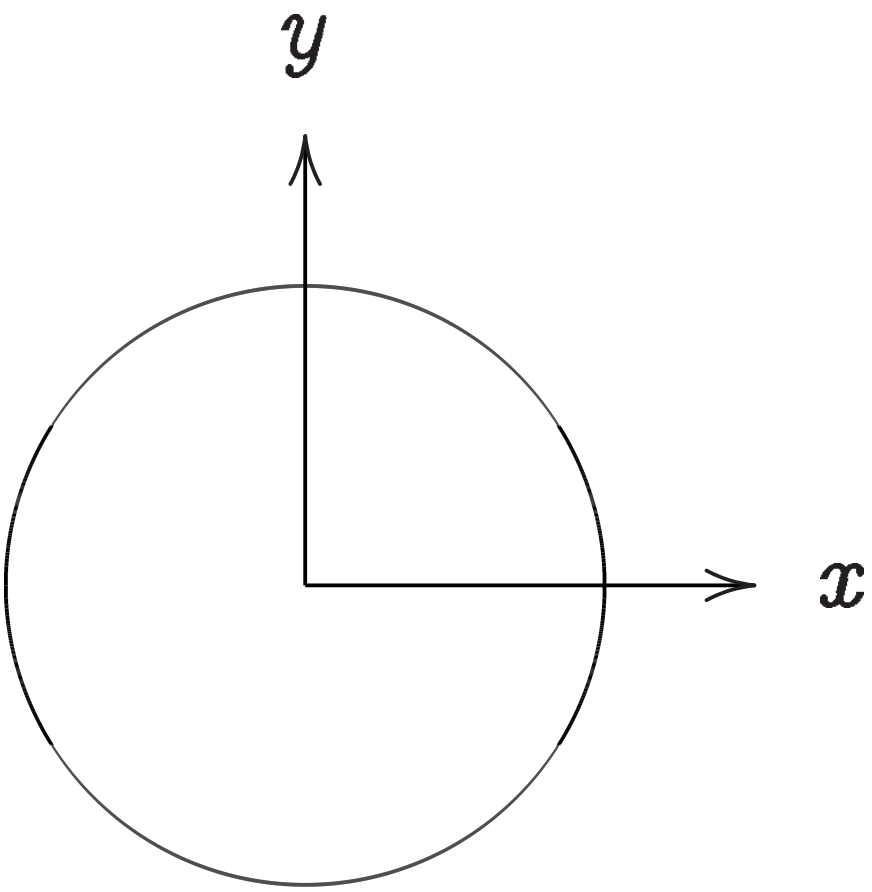
(using additive notation here).



What does P look like &

how to compute $P + Q$?

The clock



This is the curve $x^2 + y^2 = 1$.

Warning:

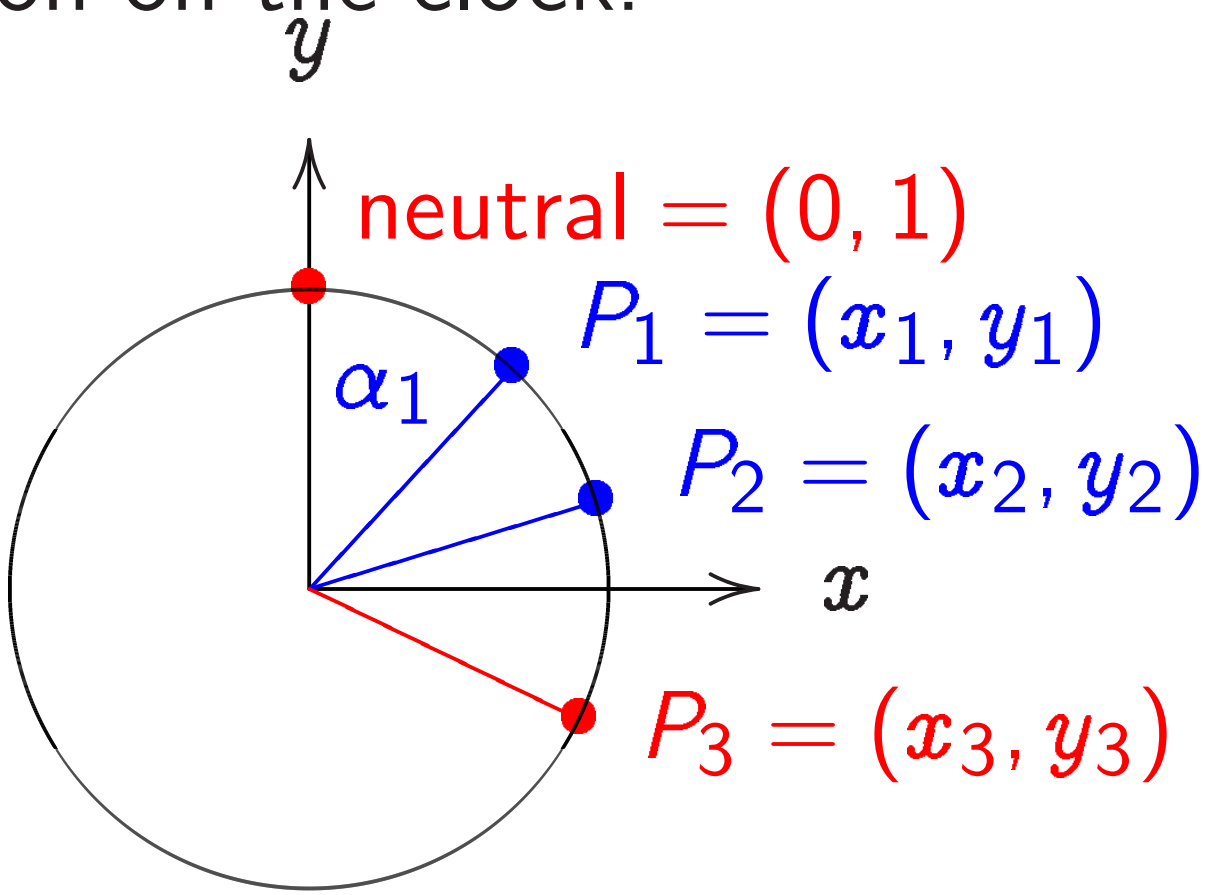
This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Adding two points corresponds
to adding the angles α_1 and α_2.
Angles modulo 360° are a group,
so points on clock are a group.

Neutral element: angle $\alpha = 0$;
point $(0, 1)$; "12:00".
The point with $\alpha = 180^\circ$
has order 2 and equals 6:00.
3:00 and 9:00 have order 4.
Inverse of point with α
is point with $-\alpha$
since $\alpha + (-\alpha) = 0$.
There are many more points
where angle α is not "nice."

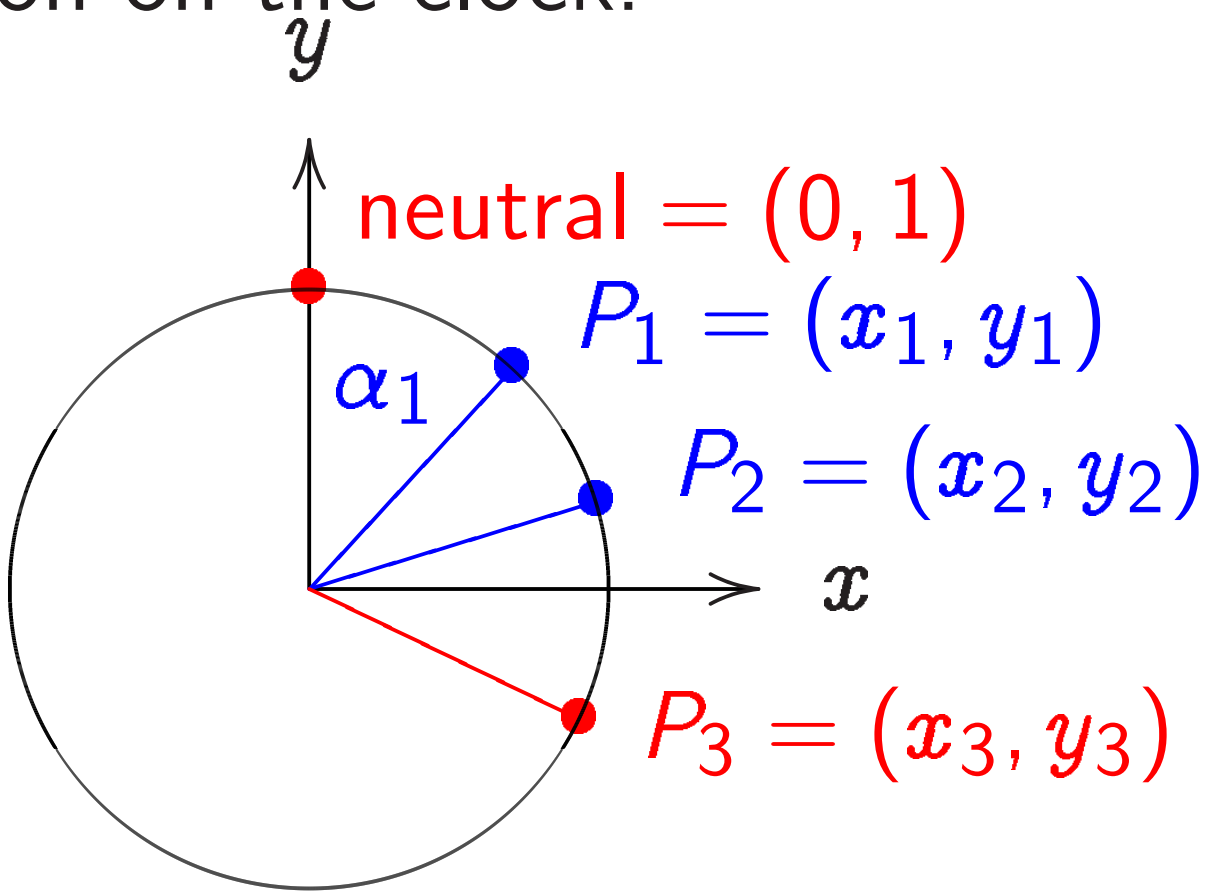
Addition on the clock:



$x^2 + y^2 = 1$, parametrized by

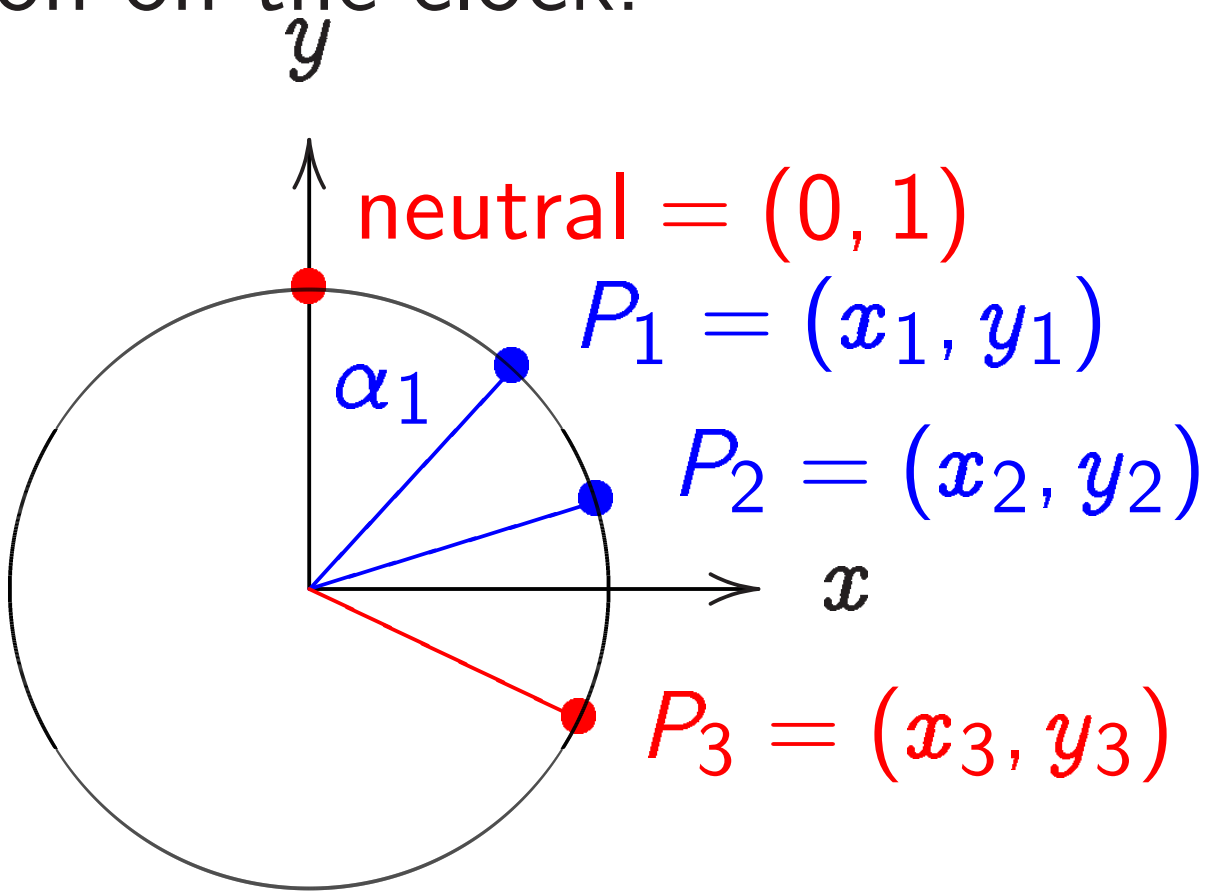
$x = \sin\alpha, \quad y = \cos\alpha.$

Addition on the clock:



$x^2 + y^2 = 1$, parametrized by $x = \sin\alpha, \quad y = \cos\alpha$. Recall $(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$

Addition on the clock:

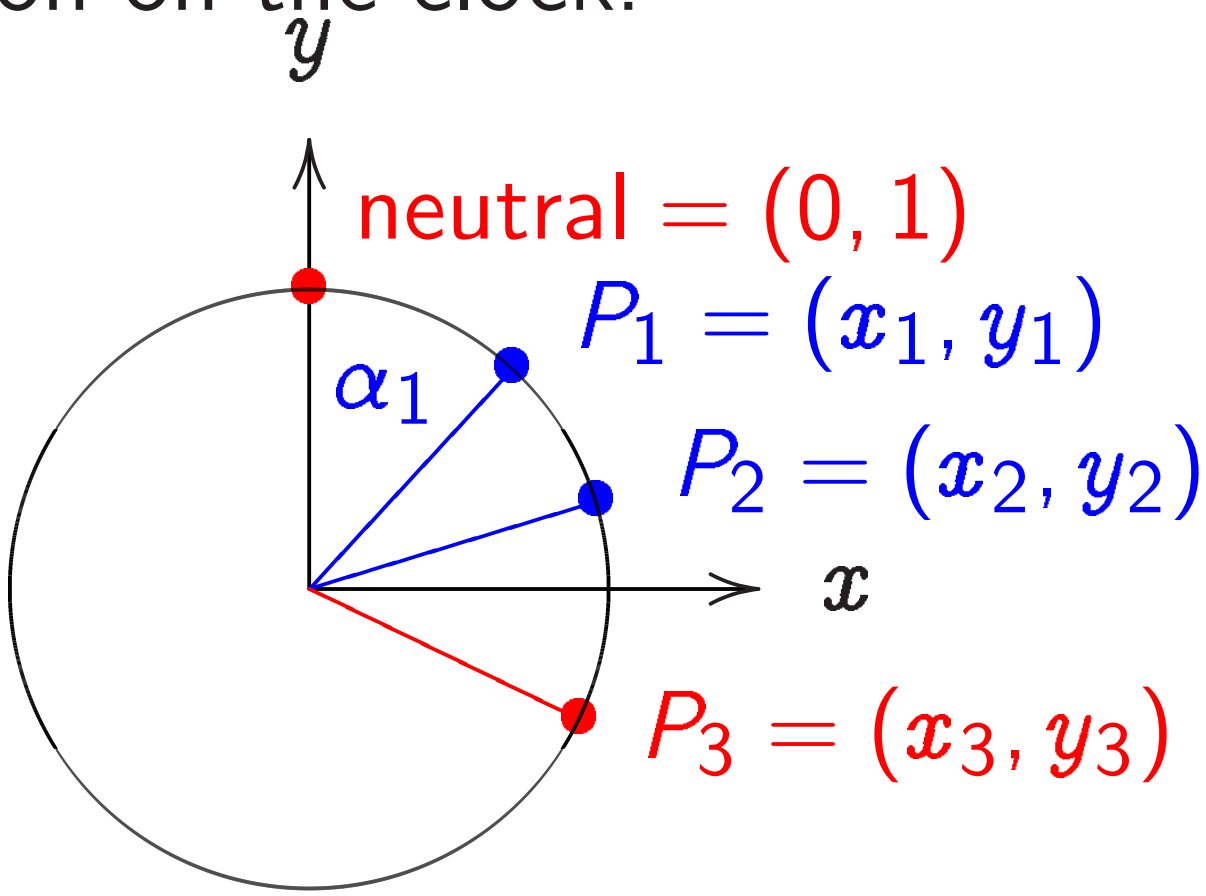


$x^2 + y^2 = 1$, parametrized by $x = \sin\alpha, \quad y = \cos\alpha$. Recall

$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$

$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$

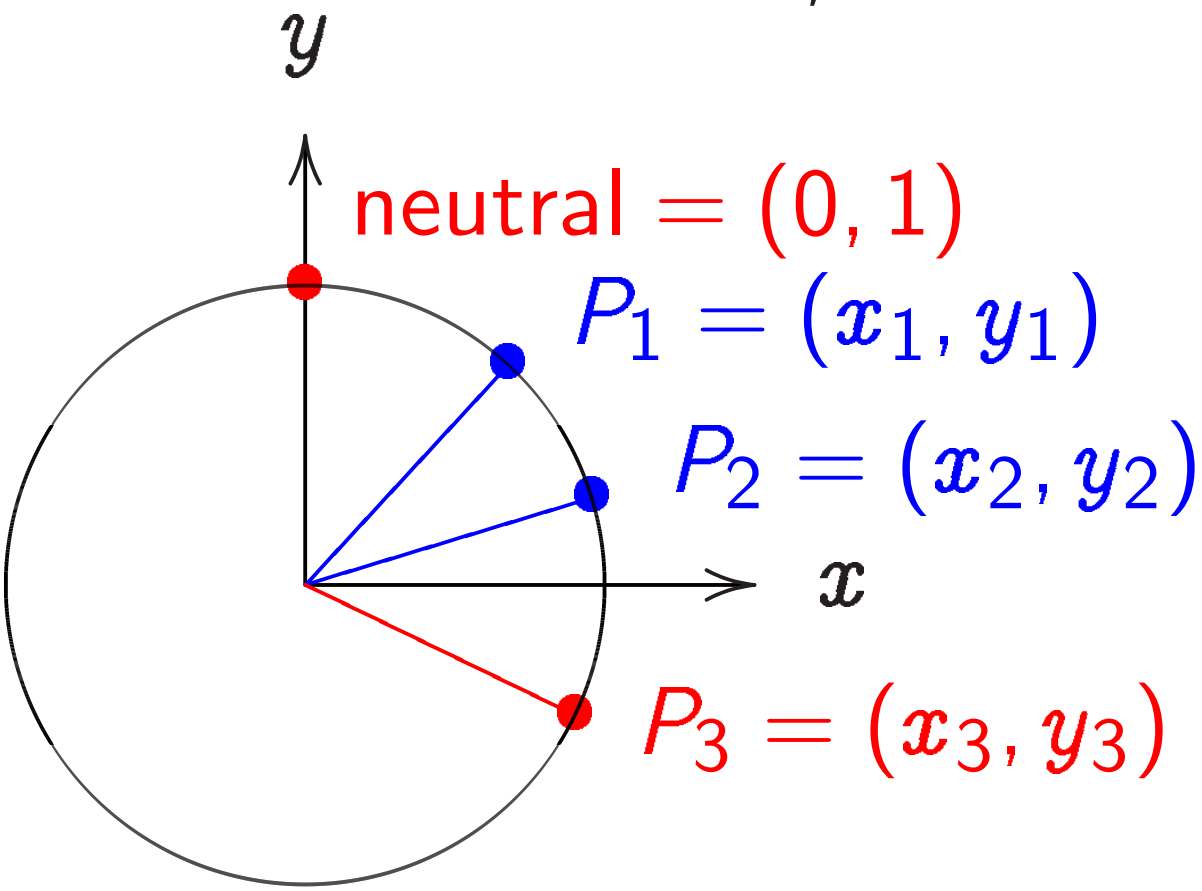
Addition on the clock:



$x^2 + y^2 = 1$, parametrized by $x = \sin\alpha, \quad y = \cos\alpha$. Recall

$$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$$
$$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$$
$$\cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2).$$

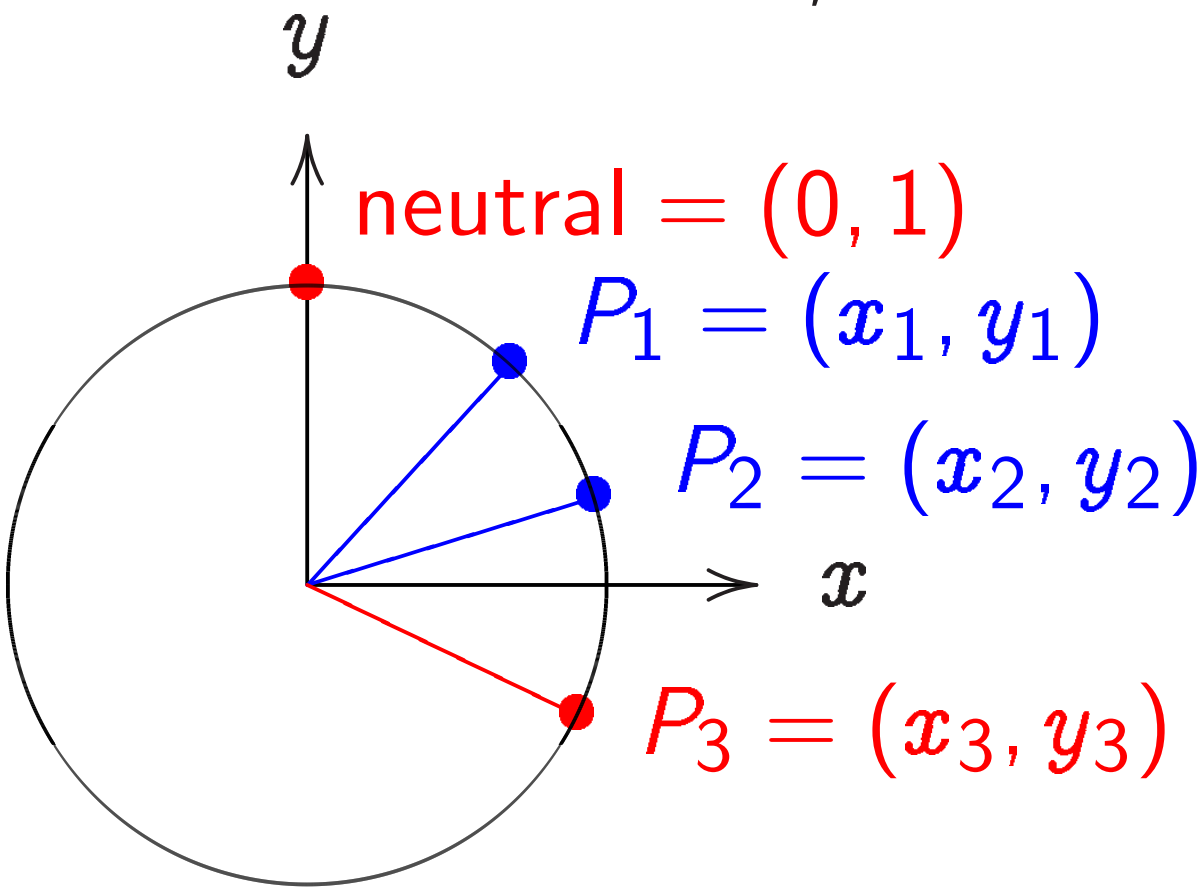
Clock addition without sin, cos:



Use Cartesian coordinates for addition. Addition formula for the clock $x^2 + y^2 = 1$:

sum $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$

Clock addition without sin, cos:



Use Cartesian coordinates for addition. Addition formula for the clock $x^2 + y^2 = 1$:

sum $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$

$= (x_1y_2 + y_1x_2, y_1y_2 - x_1x_2)$.

Note $(x_1, y_1) + (-x_1, y_1) = (0, 1)$.

$kP = \underbrace{P + P + \cdots + P}_{k \text{ copies}}$ for $k \geq 0$.

Examples of clock addition:

“2:00” + “5:00”

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ “7:00”.

“5:00” + “9:00”

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ “2:00”.

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$(x_1, y_1) + (0, 1) =$

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$(x_1, y_1) + (0, 1) = (x_1, y_1)$.

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$(x_1, y_1) + (0, 1) = (x_1, y_1).$

$(x_1, y_1) + (-x_1, y_1) =$

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$$(x_1, y_1) + (0, 1) = (x_1, y_1).$$

$$(x_1, y_1) + (-x_1, y_1) = (0, 1).$$

Clock cryptography

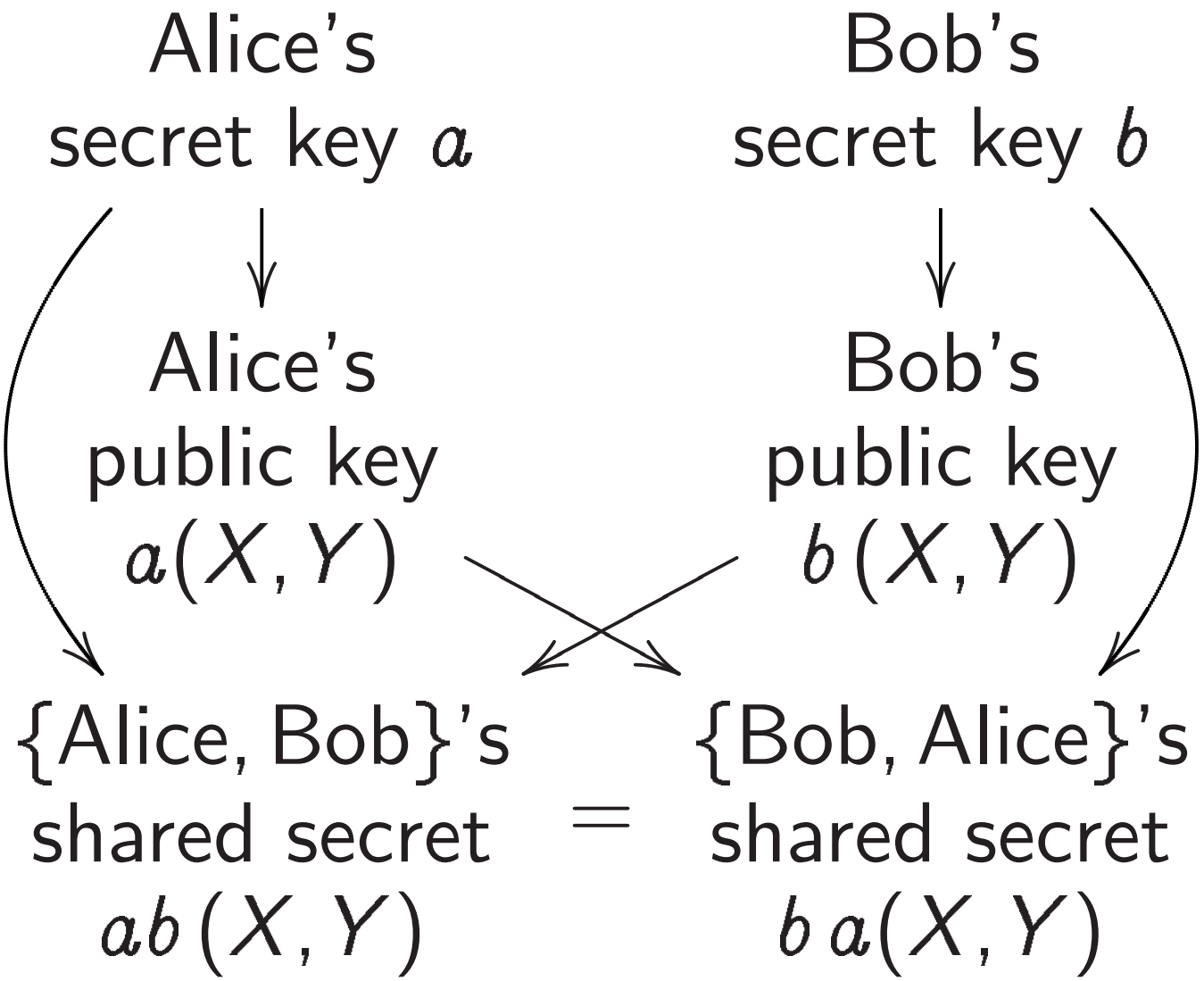
The "Clock Diffie–Hellman protocol":

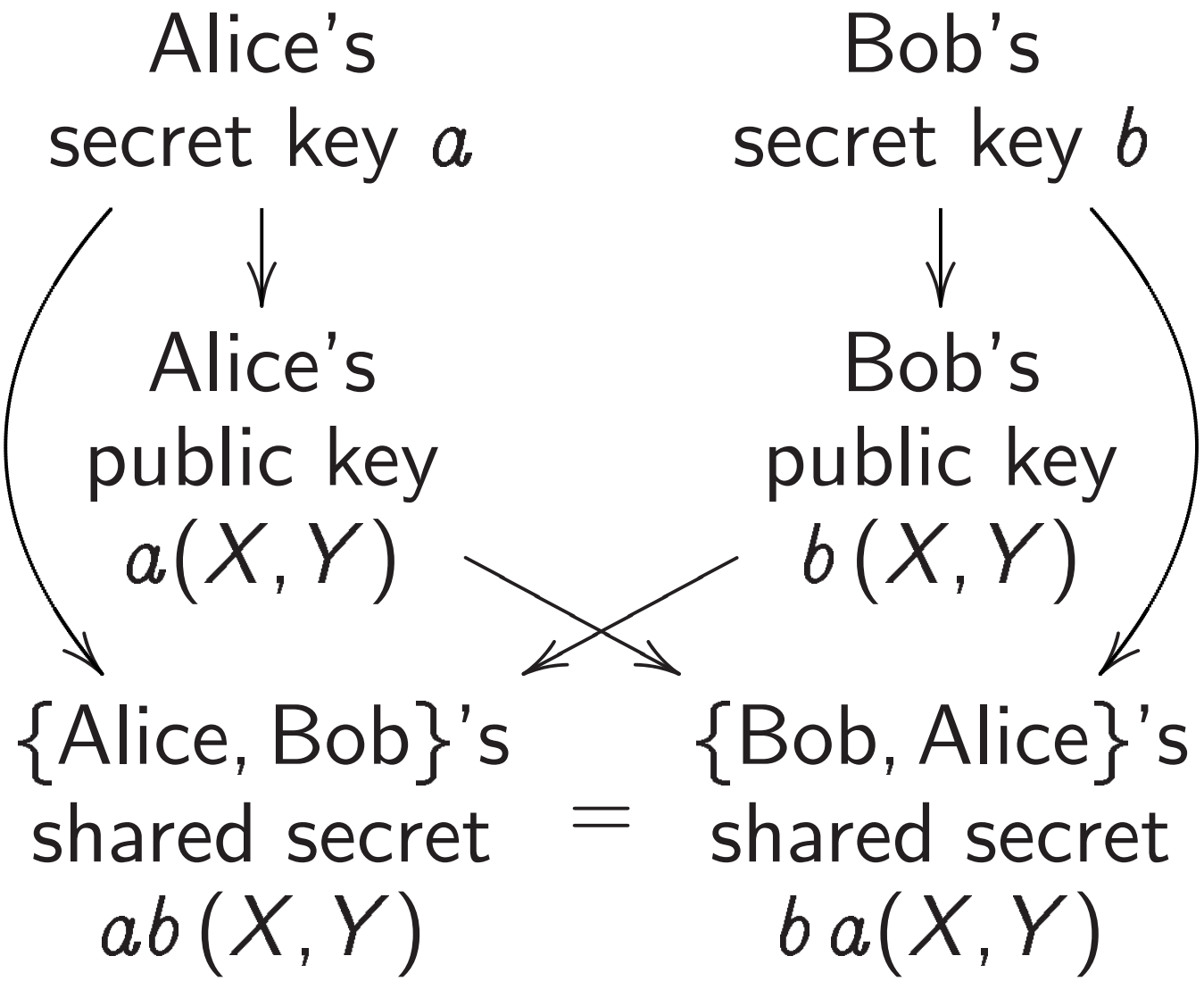
Standardize large prime p & **base point** $(x, y) \in \mathrm{Clock}(\mathbf{F}_p)$.

Alice chooses big secret a, computes her public key $a(x, y)$.

Bob chooses big secret b, computes his public key $b\,(x, y)$.

Alice computes $a(b\,(x, y))$.
Bob computes $b\,(a(x, y))$.
They use this shared secret to encrypt with AES-GCM etc.


Alice's
secret key a
Bob's
secret key b
Alice's
public key
$a(X, Y)$
Bob's
public key
$b(X, Y)$
{Alice, Bob}'s
shared secret
$ab(X, Y)$
=
{Bob, Alice}'s
shared secret
$ba(X, Y)$



Warning #1:

Many p are unsafe!

Warning #2:

Clocks aren't elliptic!

To match RSA-3072 security need $p \approx 2^{1536}$.

Warning #3:
Attacker sees more than
public keys $a(x, y)$ and $b(x, y)$.

Attacker sees how much *time*
Alice uses to compute $a(b(x, y))$.
Often attacker can see time for
each operation performed by
Alice, not just total time.
This reveals secret scalar a.

Break by timing attacks, e.g.,
2011 Brumley–Tuveri.

Warning #3:
Attacker sees more than
public keys $a(x, y)$ and $b(x, y)$.

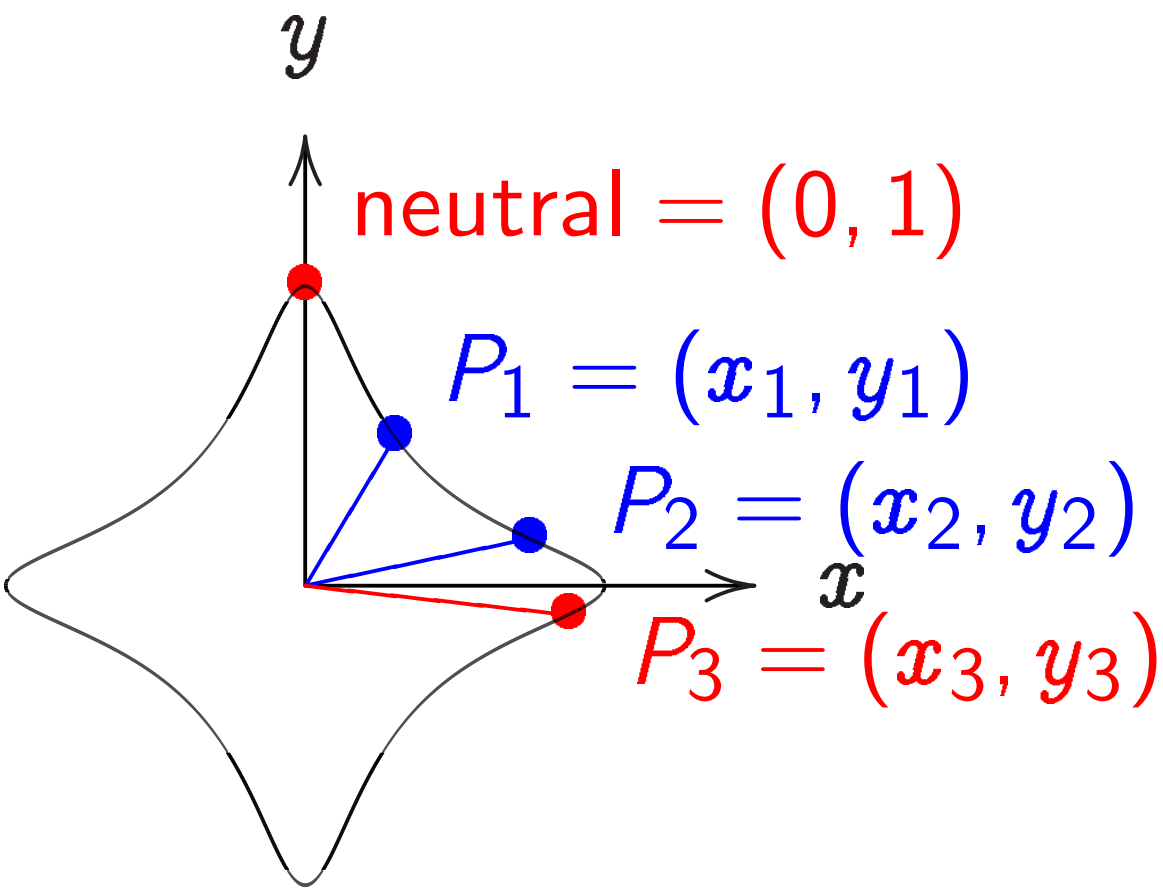
Attacker sees how much *time*
Alice uses to compute $a(b(x, y))$.
Often attacker can see time for
each operation performed by
Alice, not just total time.
This reveals secret scalar a.

Break by timing attacks, e.g.,
2011 Brumley–Tuveri.

Fix: **constant-time** code,
performing same operations
no matter what scalar is.

Addition on an Edwards curve

Change the curve on which Alice and Bob work.



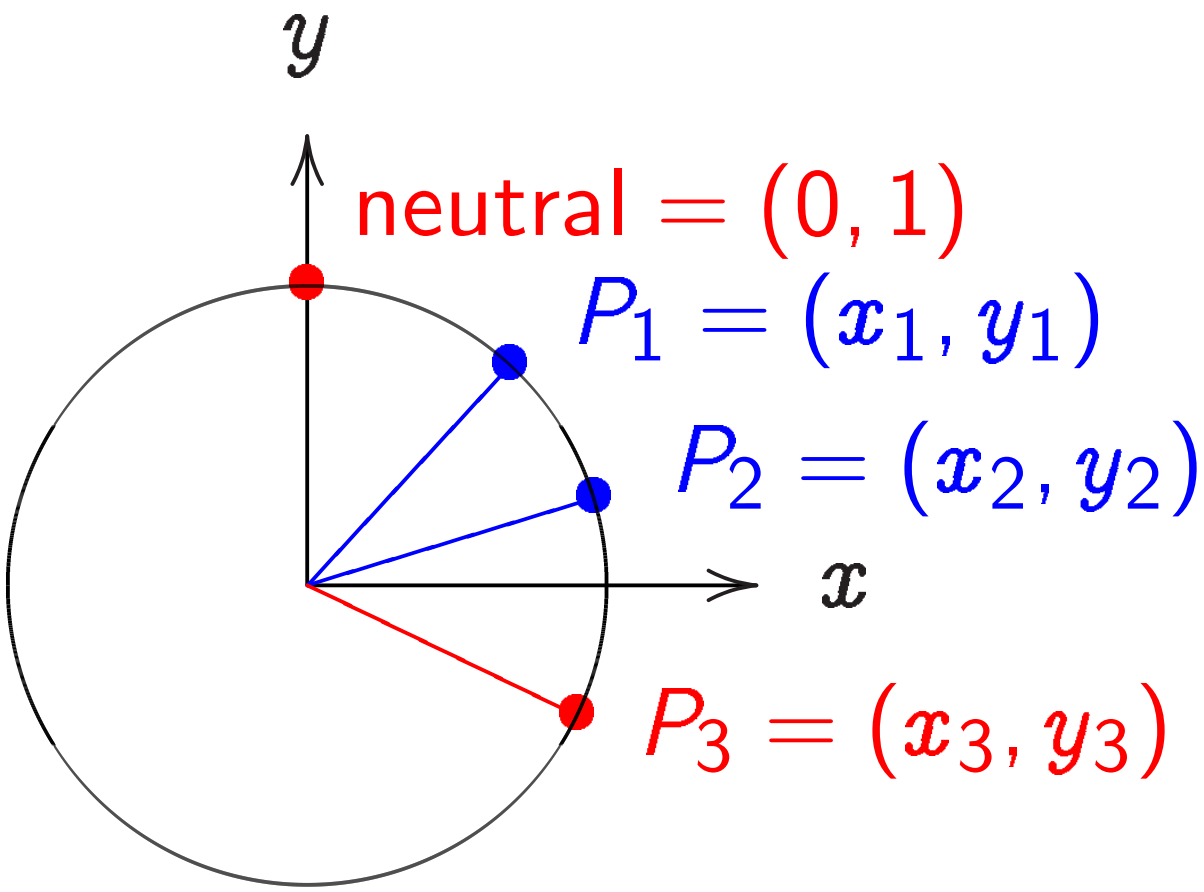
$x^2 + y^2 = 1 - 30x^2y^2$.

Sum of (x_1, y_1) and (x_2, y_2) is

$((x_1y_2+y_1x_2)/(1-30x_1x_2y_1y_2)$,

$(y_1y_2-x_1x_2)/(1+30x_1x_2y_1y_2))$.

The clock again, for comparison:



$x^2 + y^2 = 1$.

Sum of (x_1, y_1) and (x_2, y_2) is

$(x_1y_2 + y_1x_2,$

$y_1y_2 - x_1x_2)$.

"Hey, there were divisions
in the Edwards addition law!
What if the denominators are 0?"

Answer: They aren't!

If $x_i = 0$ or $y_i = 0$ then
$1 \pm 30x_1x_2y_1y_2 = 1 \neq 0$.

If $x^2 + y^2 = 1 - 30x^2y^2$
then $30x^2y^2 < 1$
so $\sqrt{30}\ |xy| < 1$.

"Hey, there were divisions
in the Edwards addition law!
What if the denominators are 0?"

Answer: They aren't!

If $x_i = 0$ or $y_i = 0$ then
$1 \pm 30x_1x_2y_1y_2 = 1 \neq 0$.

If $x^2 + y^2 = 1 - 30x^2y^2$
then $30x^2y^2 < 1$
so $\sqrt{30}\,|xy| < 1$.

If $x_1^2 + y_1^2 = 1 - 30x_1^2y_1^2$
and $x_2^2 + y_2^2 = 1 - 30x_2^2y_2^2$
then $\sqrt{30}\,|x_1y_1| < 1$
and $\sqrt{30}\,|x_2y_2| < 1$

"Hey, there were divisions
in the Edwards addition law!
What if the denominators are 0?"
Answer: They aren't!
If $x_i = 0$ or $y_i = 0$ then
$1 \pm 30x_1x_2y_1y_2 = 1 \neq 0$.
If $x^2 + y^2 = 1 - 30x^2y^2$
then $30x^2y^2 < 1$
so $\sqrt{30}\,|xy| < 1$.

If $x_1^2 + y_1^2 = 1 - 30x_1^2y_1^2$
and $x_2^2 + y_2^2 = 1 - 30x_2^2y_2^2$
then $\sqrt{30}\,|x_1y_1| < 1$
and $\sqrt{30}\,|x_2y_2| < 1$
so $30\,|x_1y_1x_2y_2| < 1$
so $1 \pm 30x_1x_2y_1y_2 > 0$.

The Edwards addition law
$(x_1, y_1) + (x_2, y_2) =$
$((x_1y_2+y_1x_2)/(1-30x_1x_2y_1y_2),$
$(y_1y_2-x_1x_2)/(1+30x_1x_2y_1y_2))$
is a group law for the curve
$x^2 + y^2 = 1 - 30x^2y^2$.

Some calculation required:
addition result is on curve;
addition law is associative.

Other parts of proof are easy:
addition law is commutative;
$(0, 1)$ is neutral element;
$(x_1, y_1) + (-x_1, y_1) = (0, 1)$.

Edwards curves mod p

Choose an odd prime p.

Choose a *non-square* $d \in \mathbf{F}_p$.

$\{(x, y) \in \mathbf{F}_p \times \mathbf{F}_p :$
$x^2 + y^2 = 1 + dx^2y^2\}$
is a "complete Edwards curve".

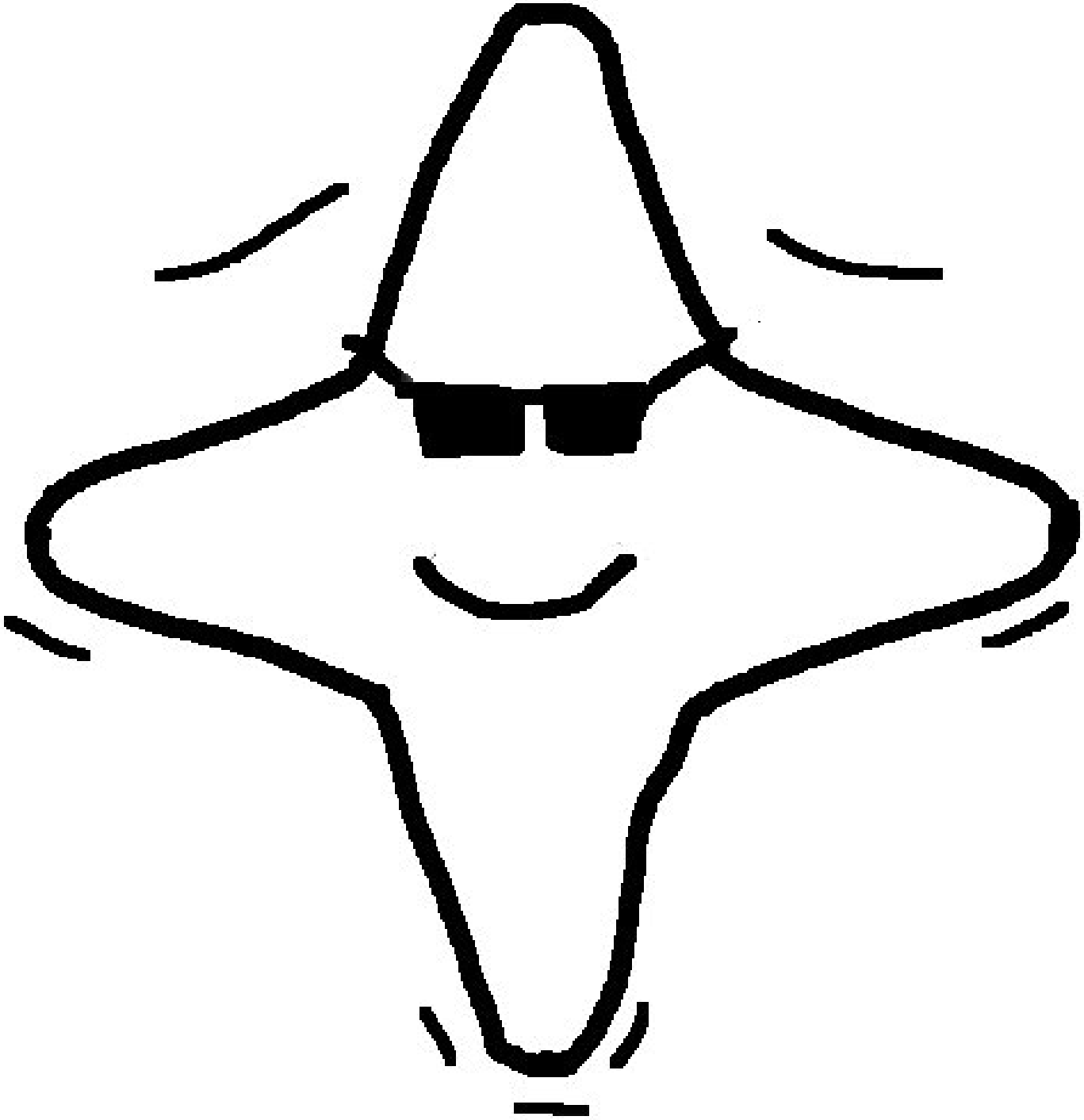
Roughly $p + 1$ pairs (x, y).

Denominators are never 0.

But proof is different;
"$x^2 + y^2 > 0$" doesn't work.

This proof relies on
choosing *non-square* d.

Edwards curves are cool



More elliptic curves

Edwards curves are elliptic.
Easiest way to understand
elliptic curves is Edwards.

Geometrically, all elliptic curves
are Edwards curves.

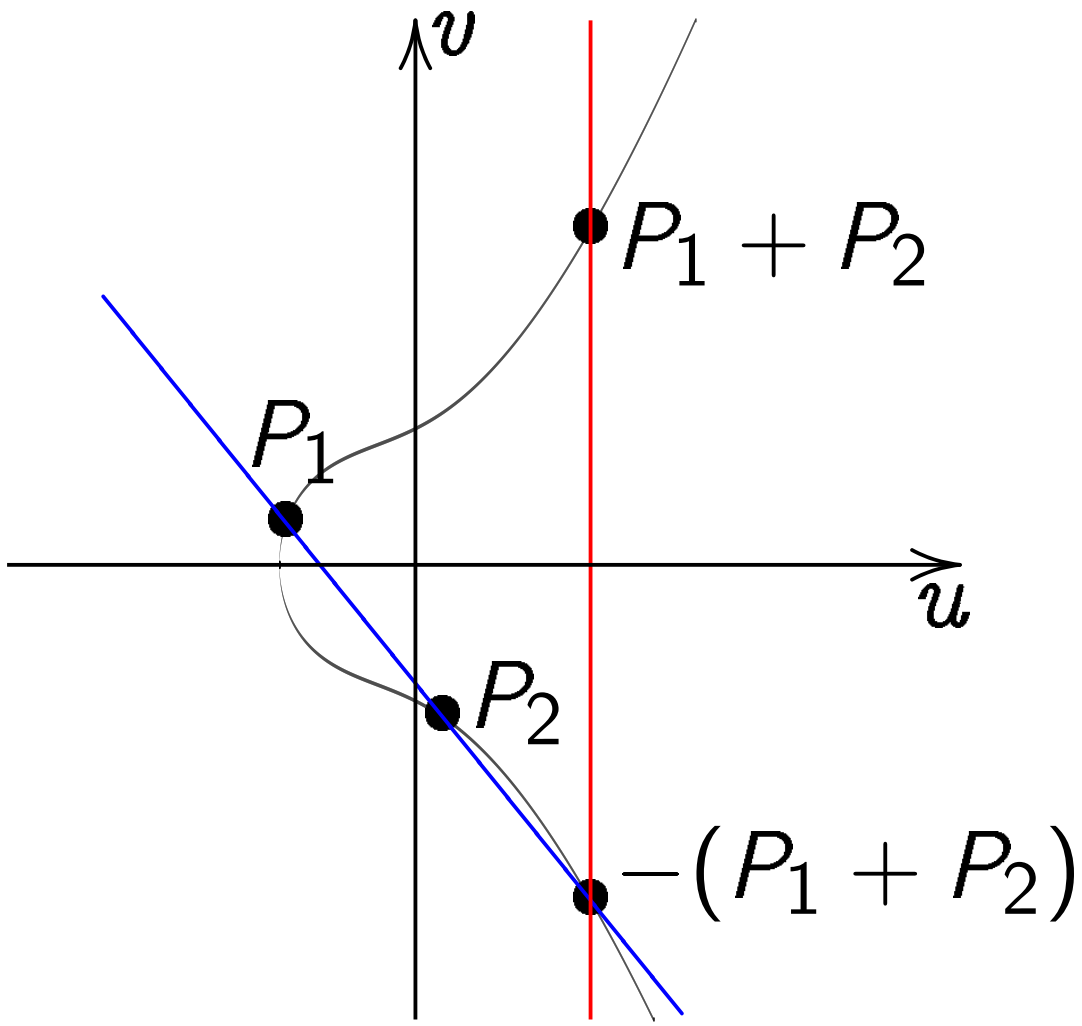
Algebraically,
more elliptic curves exist
(not always point of order 4).

Every odd-char curve can be
expressed as Weierstrass curve
$v^2 = u^3 + a_2u^2 + a_4u + a_6$.

Warning: "Weierstrass" has
different meaning in char 2.

Addition on Weierstrass curve

$$v^2 = u^3 + u^2 + u + 1$$

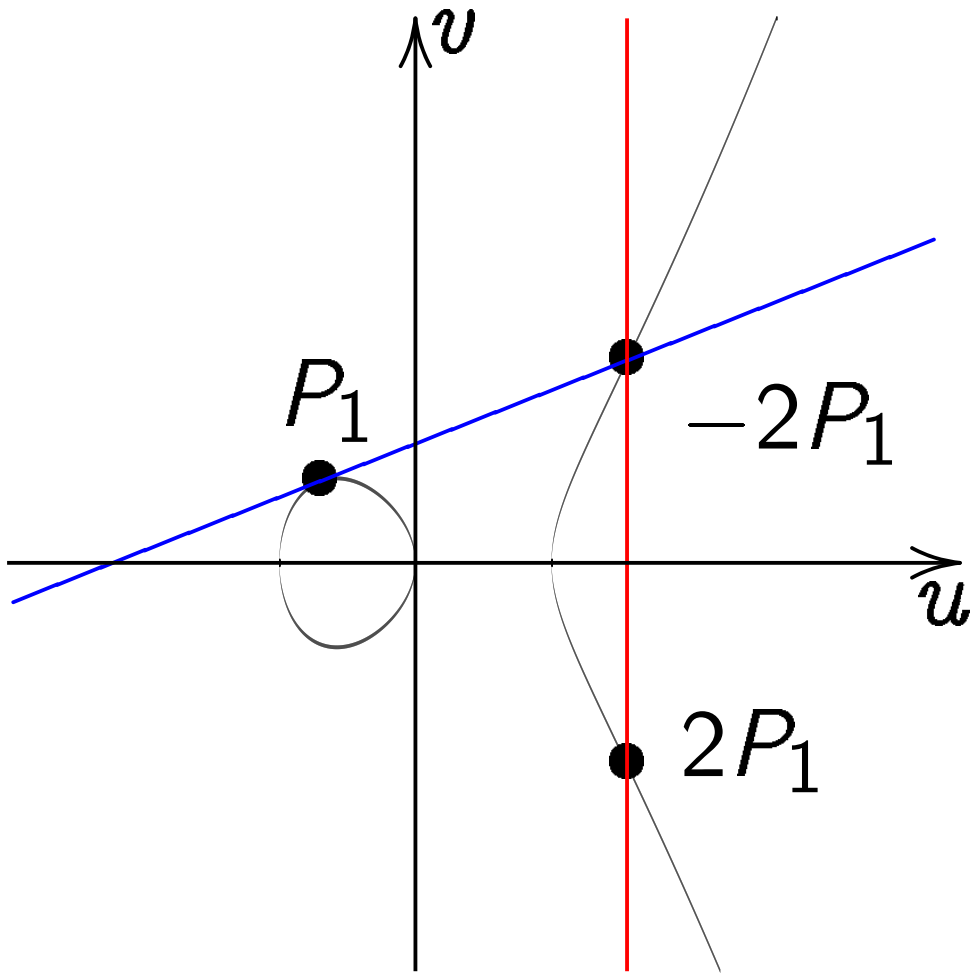


Slope $\lambda = (v_2 - v_1)/(u_2 - u_1)$.

Note that $u_1 \neq u_2$. Some points missing, in particular ∞.

Doubling on Weierstrass curve

$v^2 = u^3 - u$



Slope $\lambda = (3u_1^2 - 1)/(2v_1)$.

In most cases
$(u_1, v_1) + (u_2, v_2) =$
(u_3, v_3) where $(u_3, v_3) =$
$(\lambda^2 - u_1 - u_2, \lambda(u_1 - u_3) - v_1)$.

$u_1 \neq u_2$, "addition" (alert!):
$\lambda = (v_2 - v_1)/(u_2 - u_1)$.
Total cost $1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$.

$(u_1, v_1) = (u_2, v_2)$ and $v_1 \neq 0$,
"doubling" (alert!):
$\lambda = (3u_1^2 + 2a_2u_1 + a_4)/(2v_1)$.
Total cost $1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$.

Also handle some exceptions:
$(u_1, v_1) = (u_2, -v_2)$; ∞ as input.
Messy to implement and test.

Birational equivalence

Starting from point (x, y)
on $x^2 + y^2 = 1 + dx^2y^2$:

Define $A = 2(1 + d)/(1 - d)$,
$B = 4/(1 - d)$;
$u = (1 + y)/(B(1 - y))$,
$v = u/x = (1 + y)/(Bx(1 - y))$.
(Skip a few exceptional points.)

Then (u, v) is a point on
a Weierstrass curve:
$v^2 = u^3 + (A/B)u^2 + (1/B^2)u$.

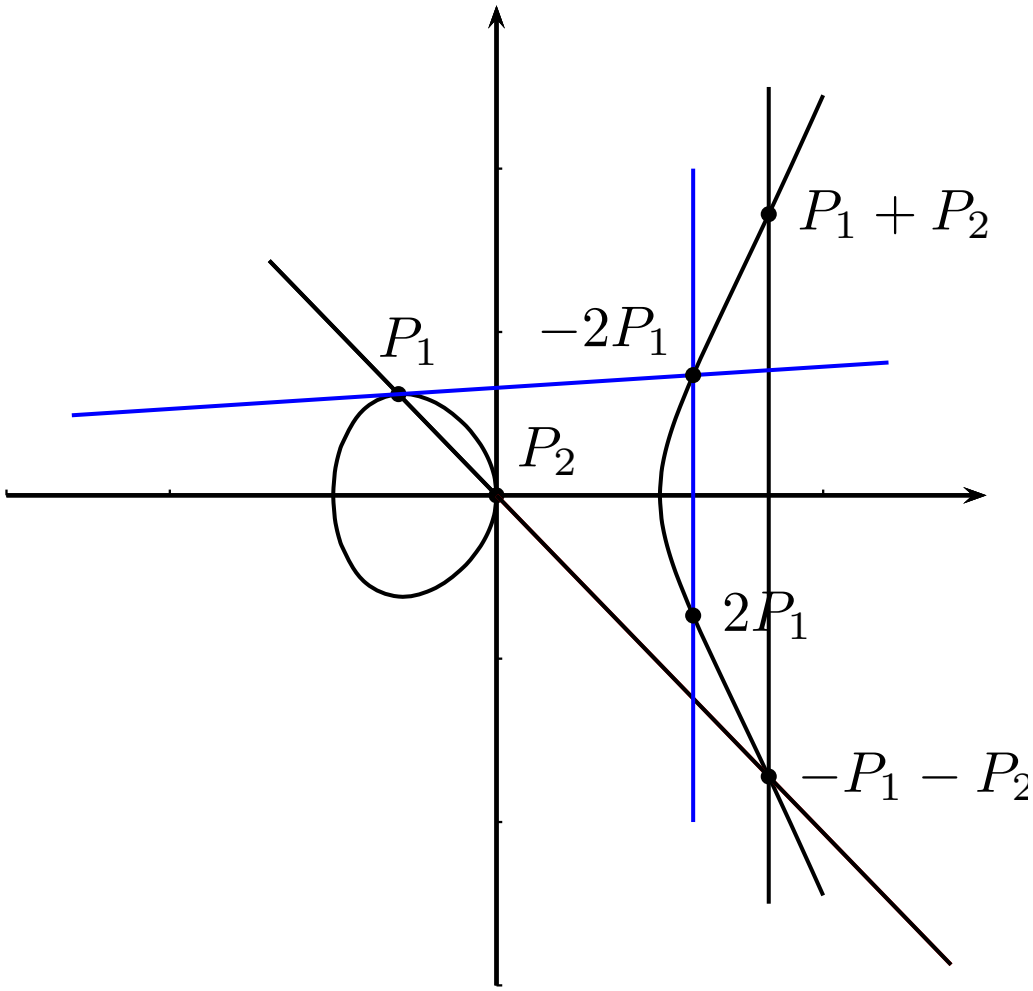
Easily invert this map:
$x = u/v$, $y = (Bu - 1)/(Bu + 1)$.

Attacker can transform Edwards curve to Weierstrass curve and vice versa; $n(x, y) \mapsto n(u, v)$.
$\Rightarrow$ Same hardness of finding n!
Can choose curve representation so that implementation of attack is faster/easier.

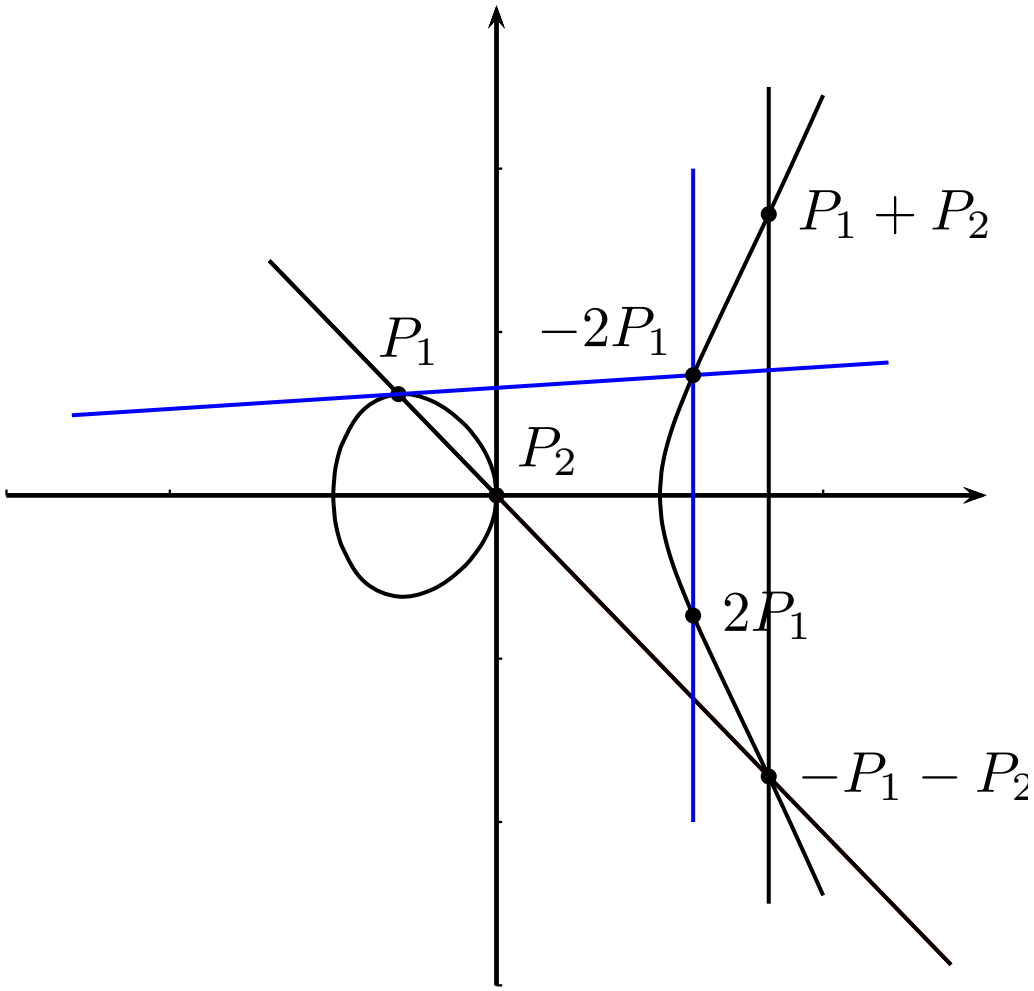
System designer can choose curve representation so that protocol runs fastest; no need to worry about security degradation.

Optimization targets are different.

Elliptic-curve groups



Elliptic-curve groups



Following algorithms will need a unique representative per point. For that, Weierstrass curves are the speed leader.

The discrete-logarithm problem

Define $p = 1000003$ and consider the Weierstrass curve $y^2 = x^3 - x$ over $\mathbf{F}_p$.

This curve has

The discrete-logarithm problem

Define $p = 1000003$ and consider the Weierstrass curve $y^2 = x^3 - x$ over $\mathbf{F}_p$.

This curve has $1000004 = 2^2 \cdot 53^2 \cdot 89$ points and $P = (101384, 614510)$ is a point of order $2 \cdot 53^2 \cdot 89$.

The discrete-logarithm problem

Define $p = 1000003$ and consider the Weierstrass curve $y^2 = x^3 - x$ over $\mathbf{F}_p$.

This curve has $1000004 = 2^2 \cdot 53^2 \cdot 89$ points and $P = (101384, 614510)$ is a point of order $2 \cdot 53^2 \cdot 89$.

In general, point counting over $\mathbf{F}_p$ runs in time polynomial in $\log p$.

Number of points in $[p + 1 - 2\sqrt{p}, p + 1 + 2\sqrt{p}]$.

The group is isomorphic to $\mathbf{Z}/k \times \mathbf{Z}/m$, where $k|m$ and $k|(p-1)$.

Can we find an integer
$n \in \{1, 2, 3, \ldots, 500001\}$
such that $nP = (670366, 740819)$?

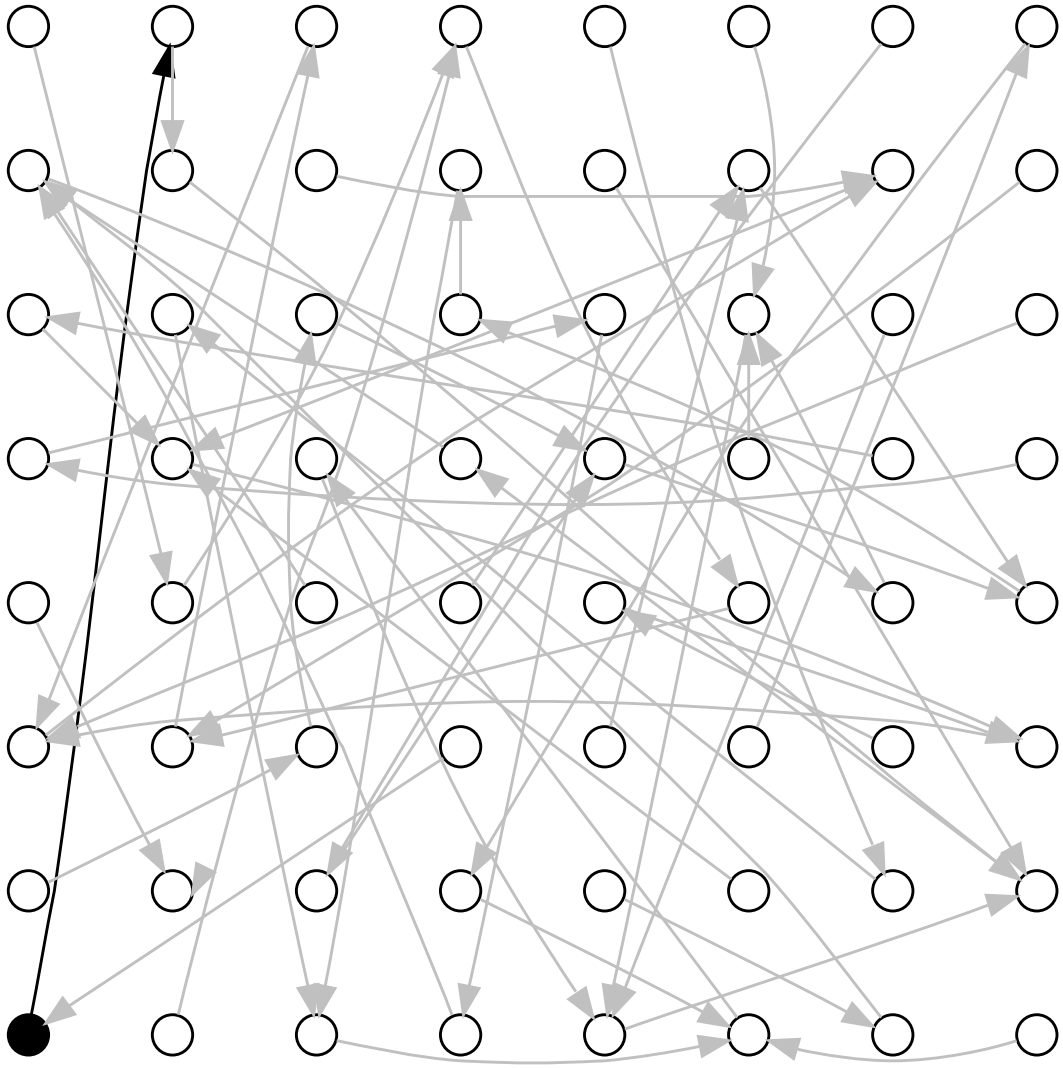
This point was generated as a multiple of P, so DL is valid.

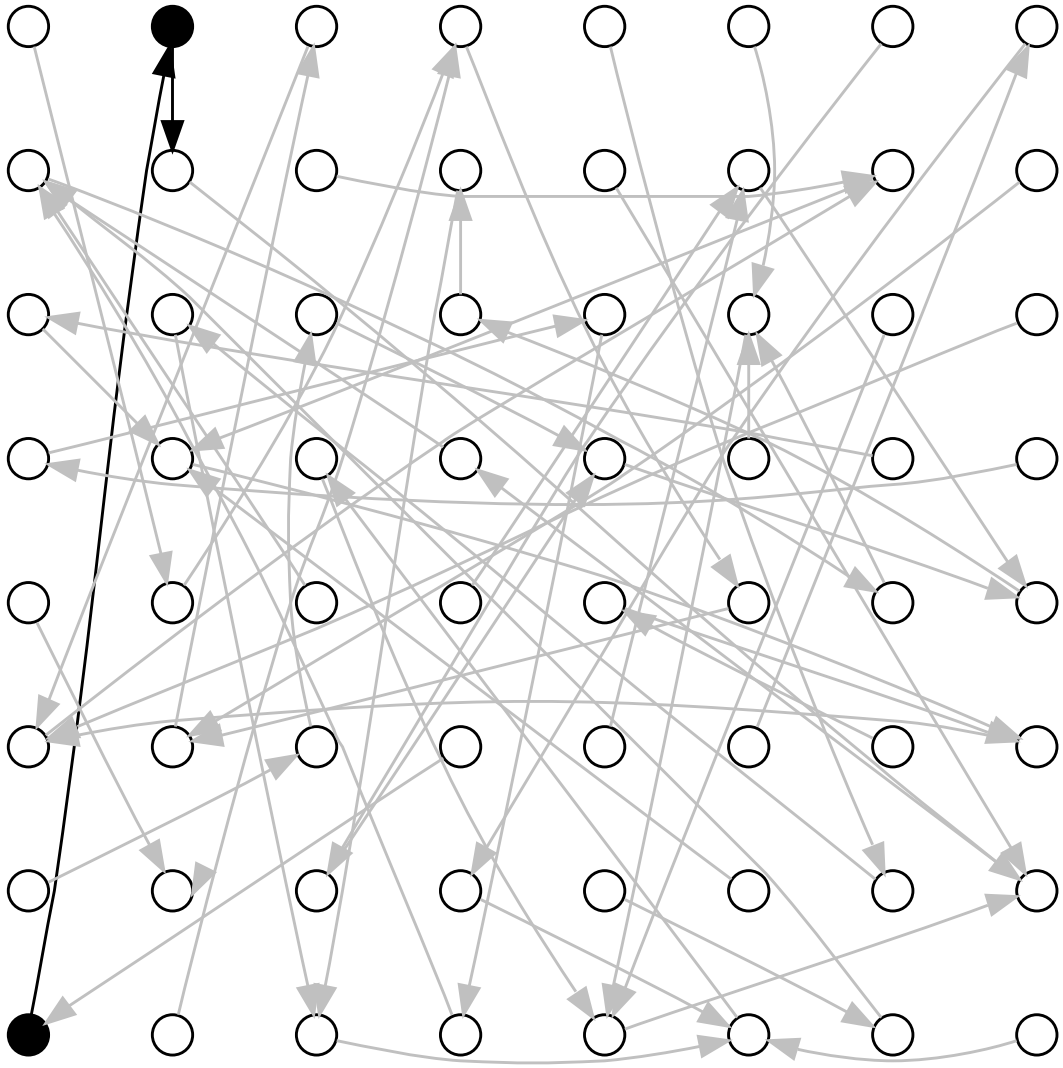
Could find n by brute force.
Is there a faster way?

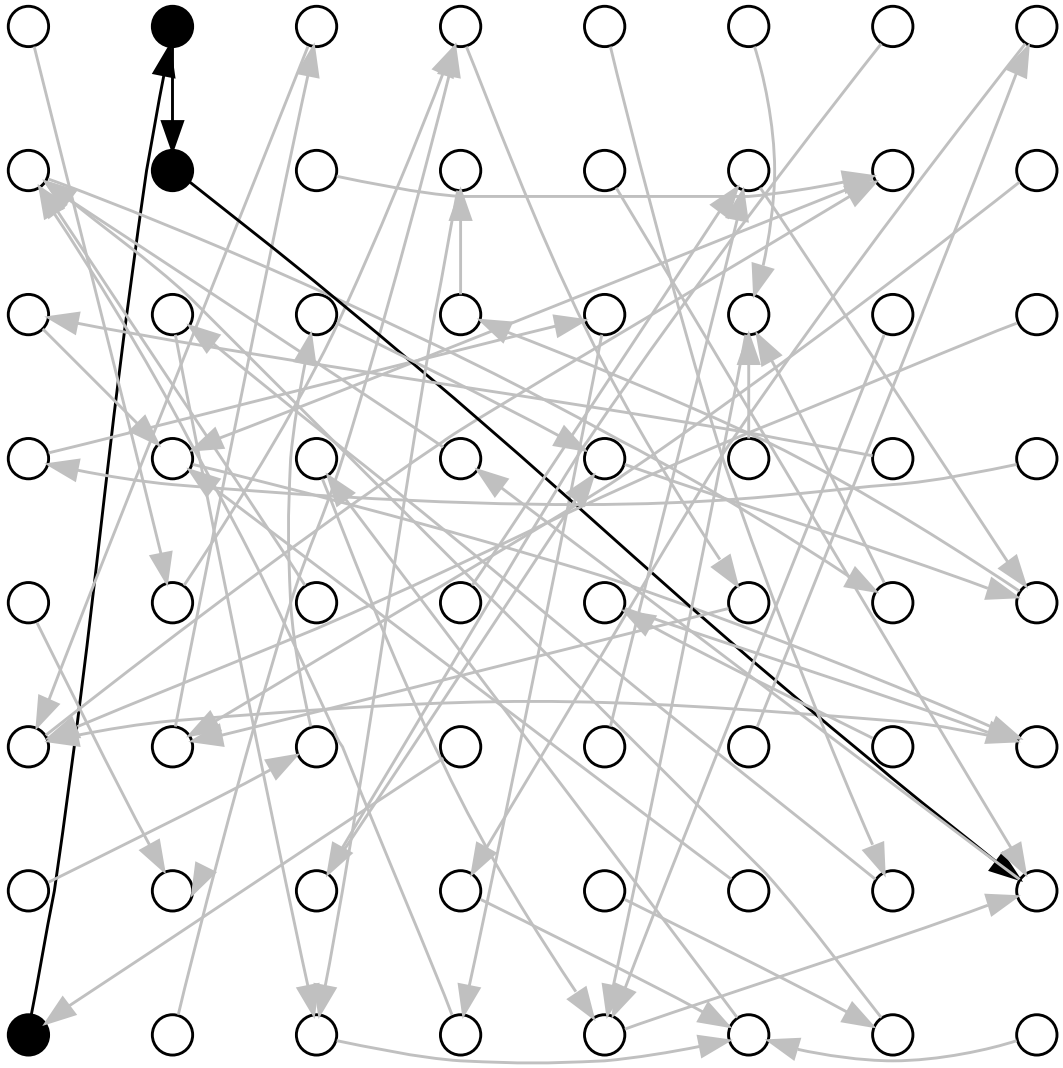
The rho method

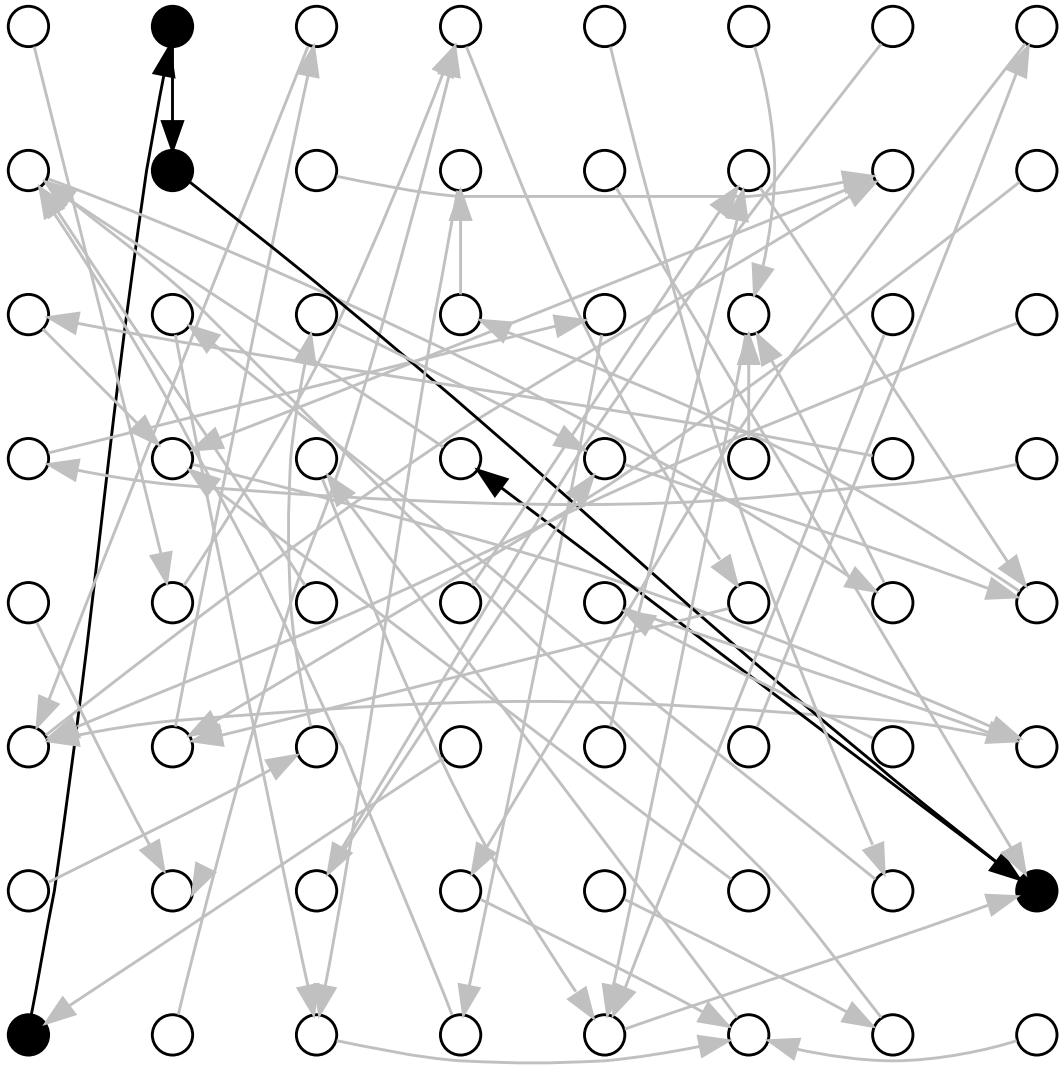
Simplified, non-parallel rho:

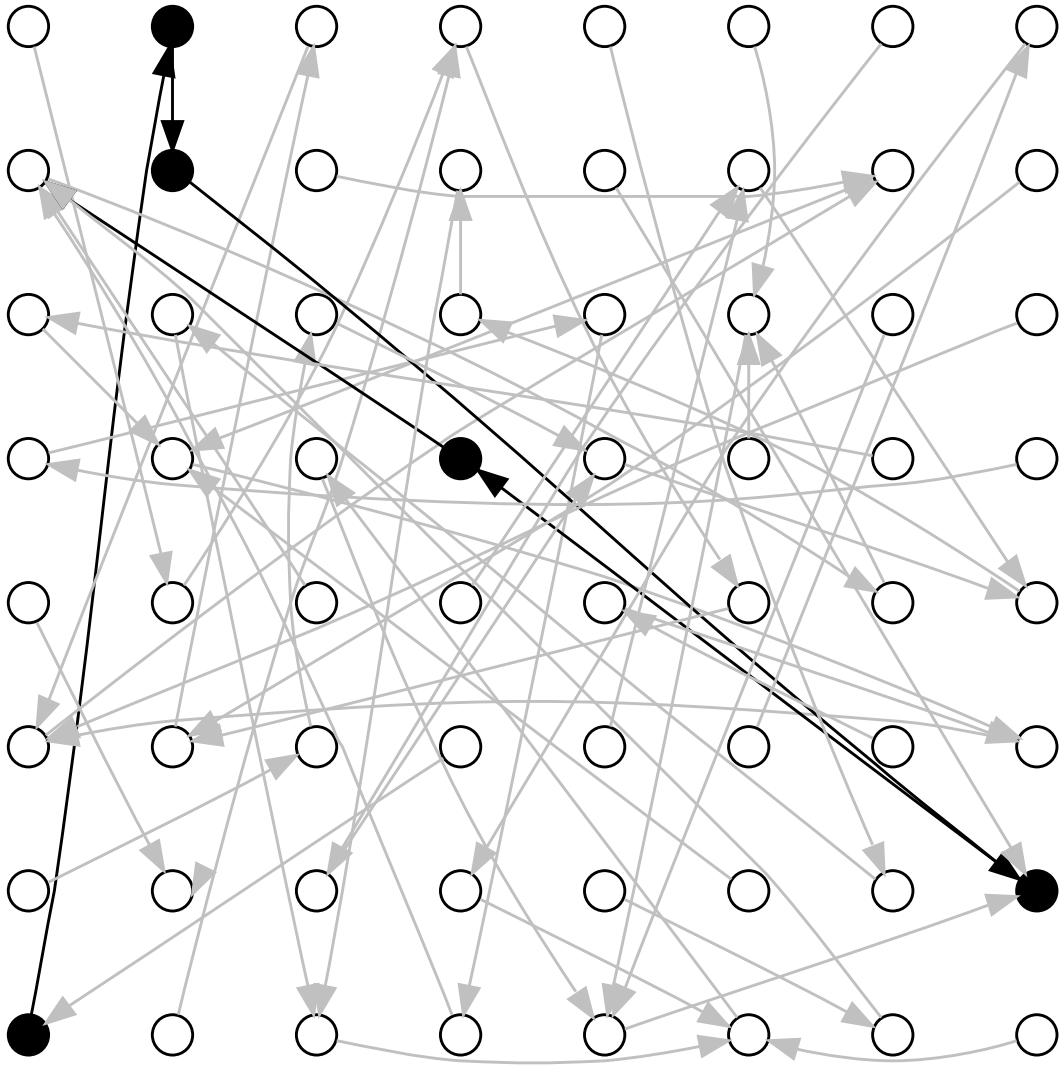
Make a pseudo-random walk
in the group $\langle P \rangle$,
where the next step depends
on current point: $W_{i+1} = f(W_i)$.

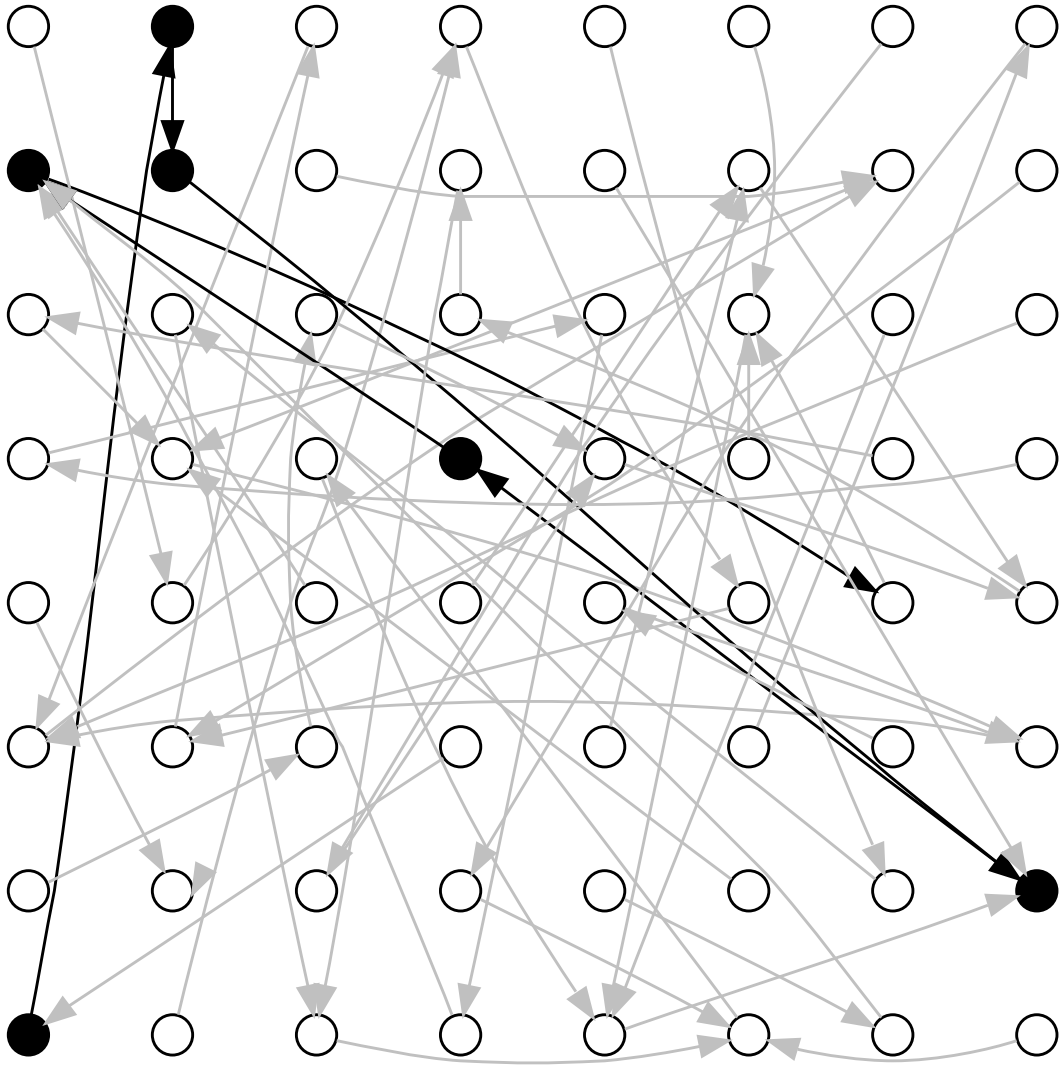


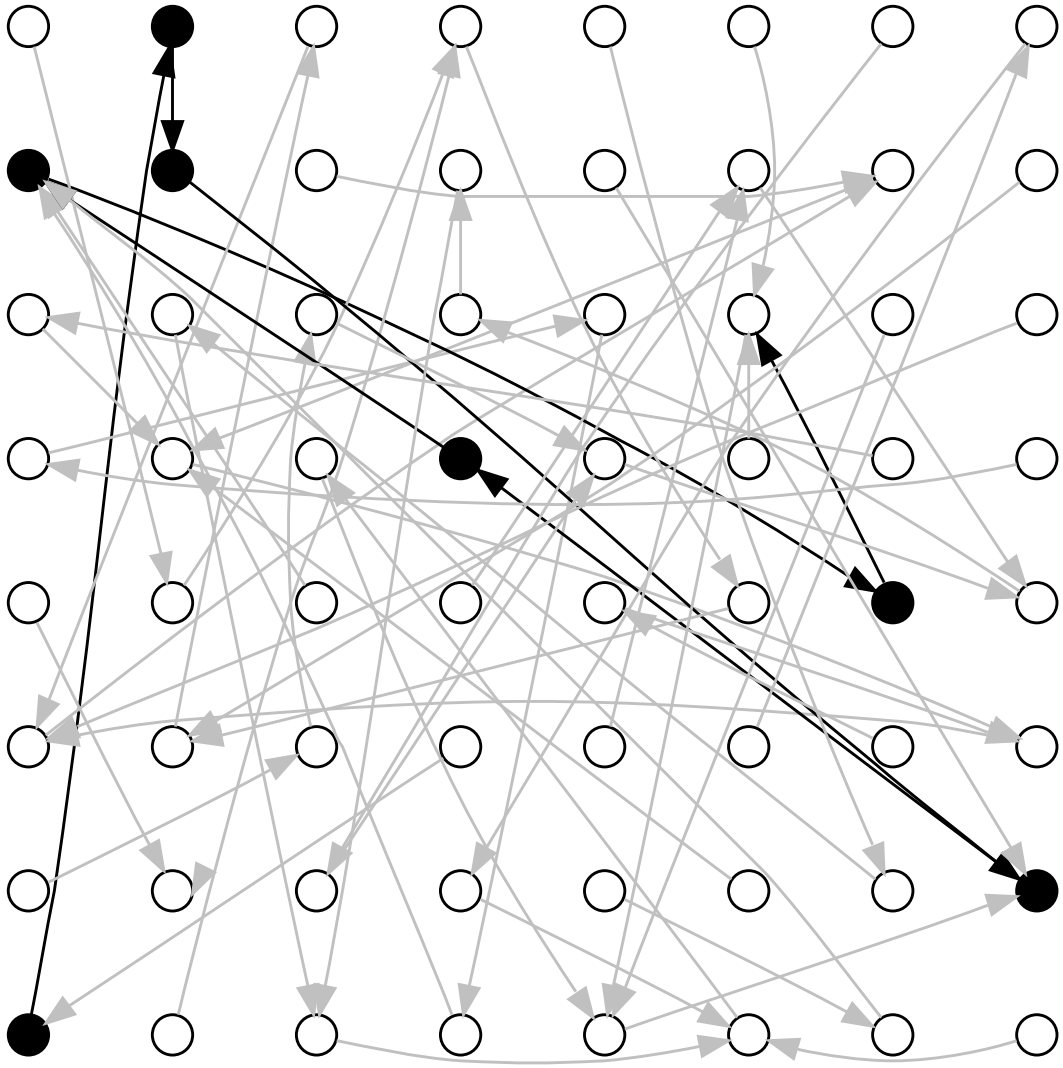


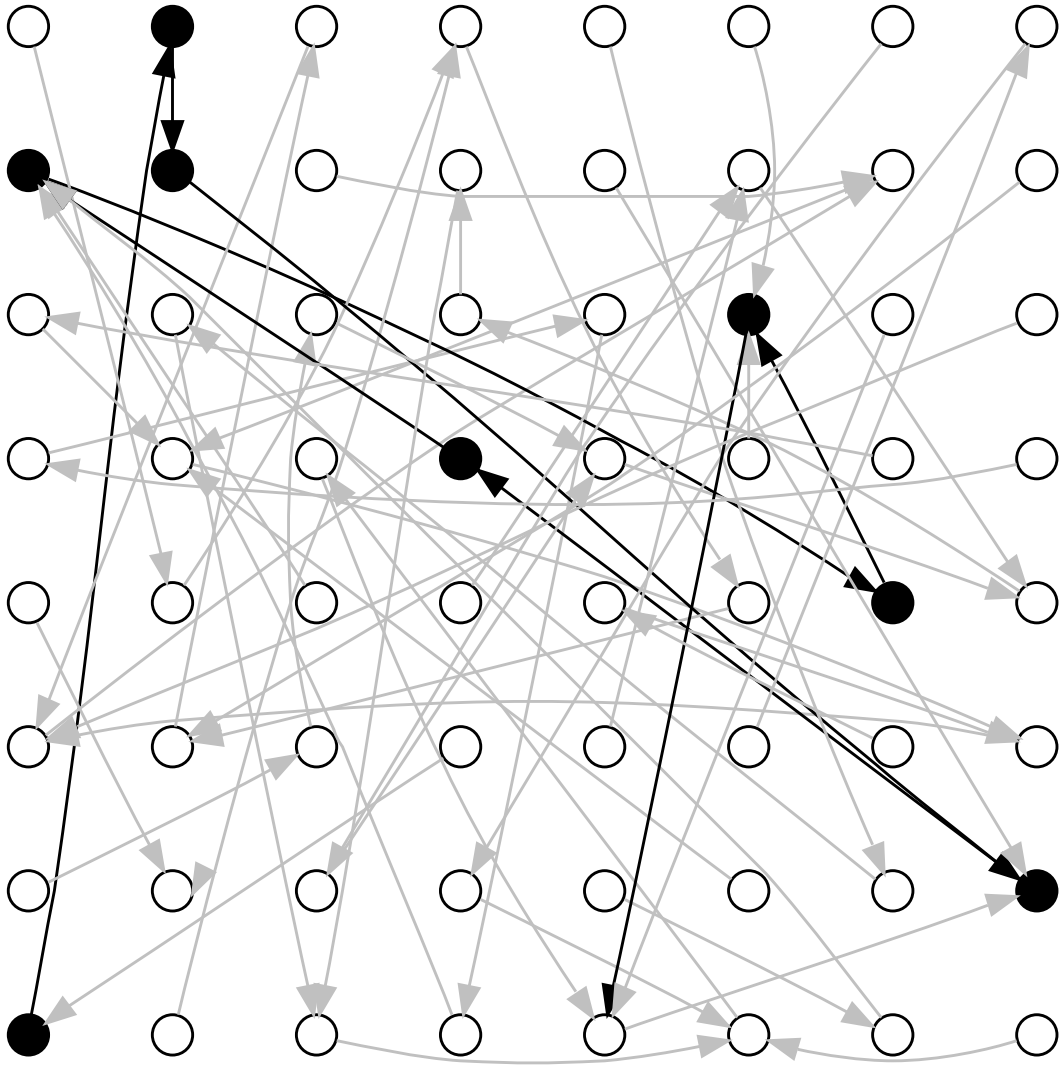


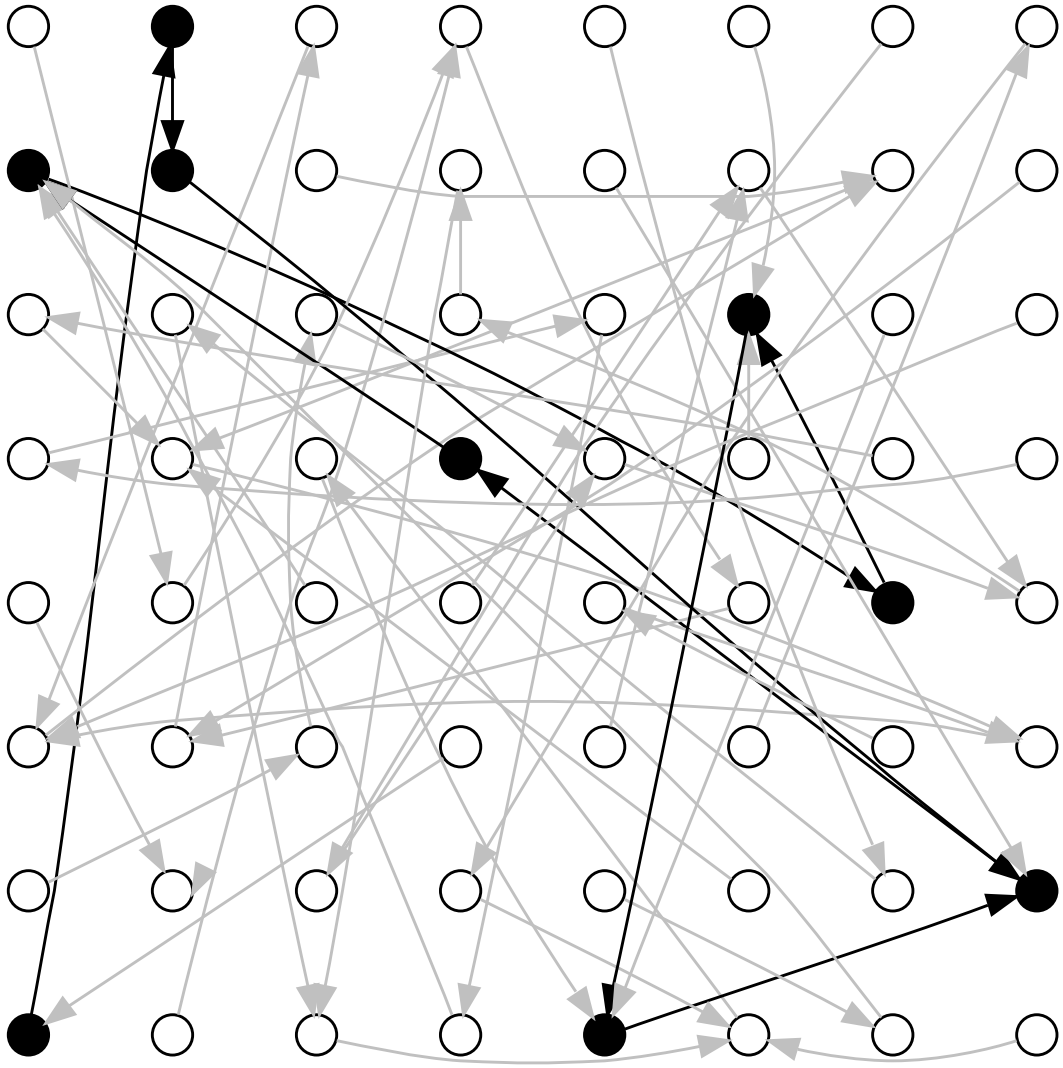


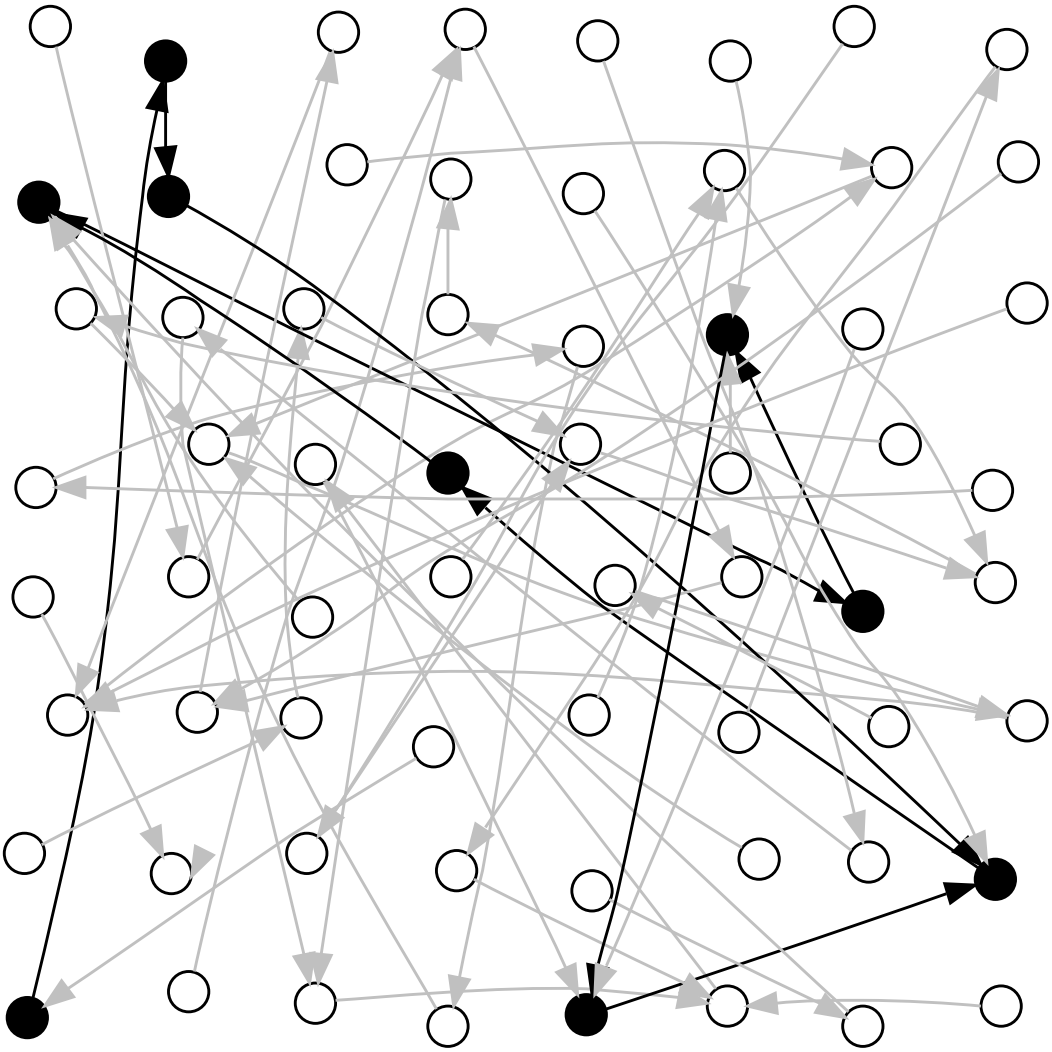


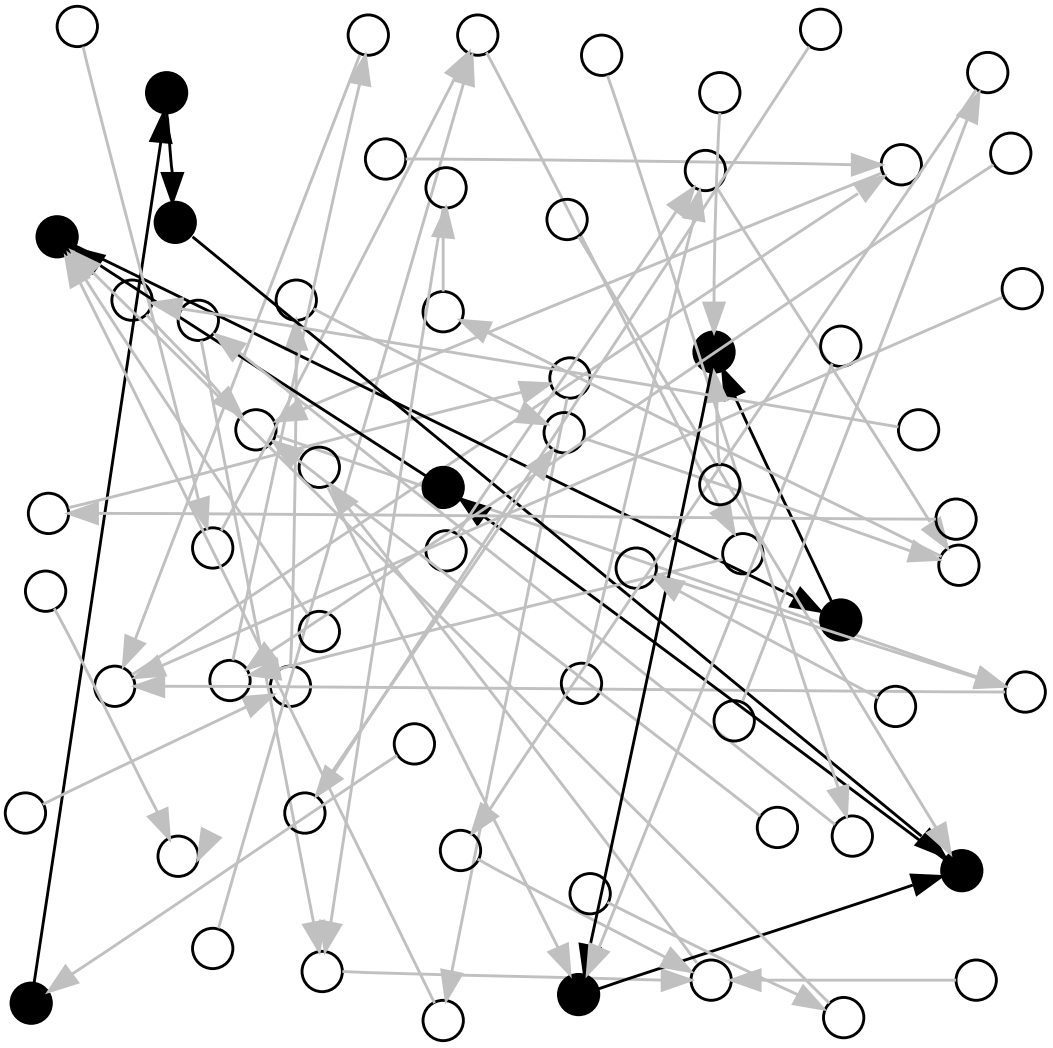


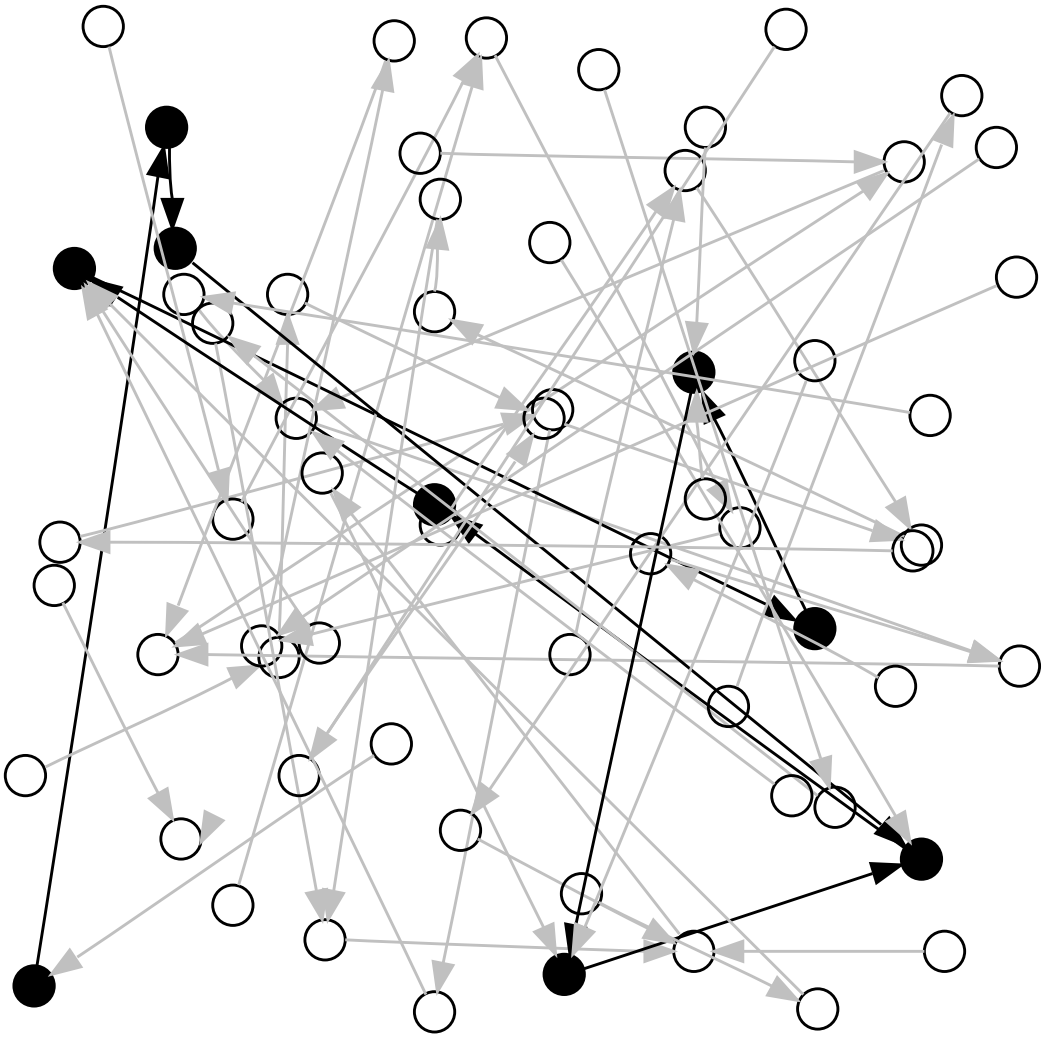


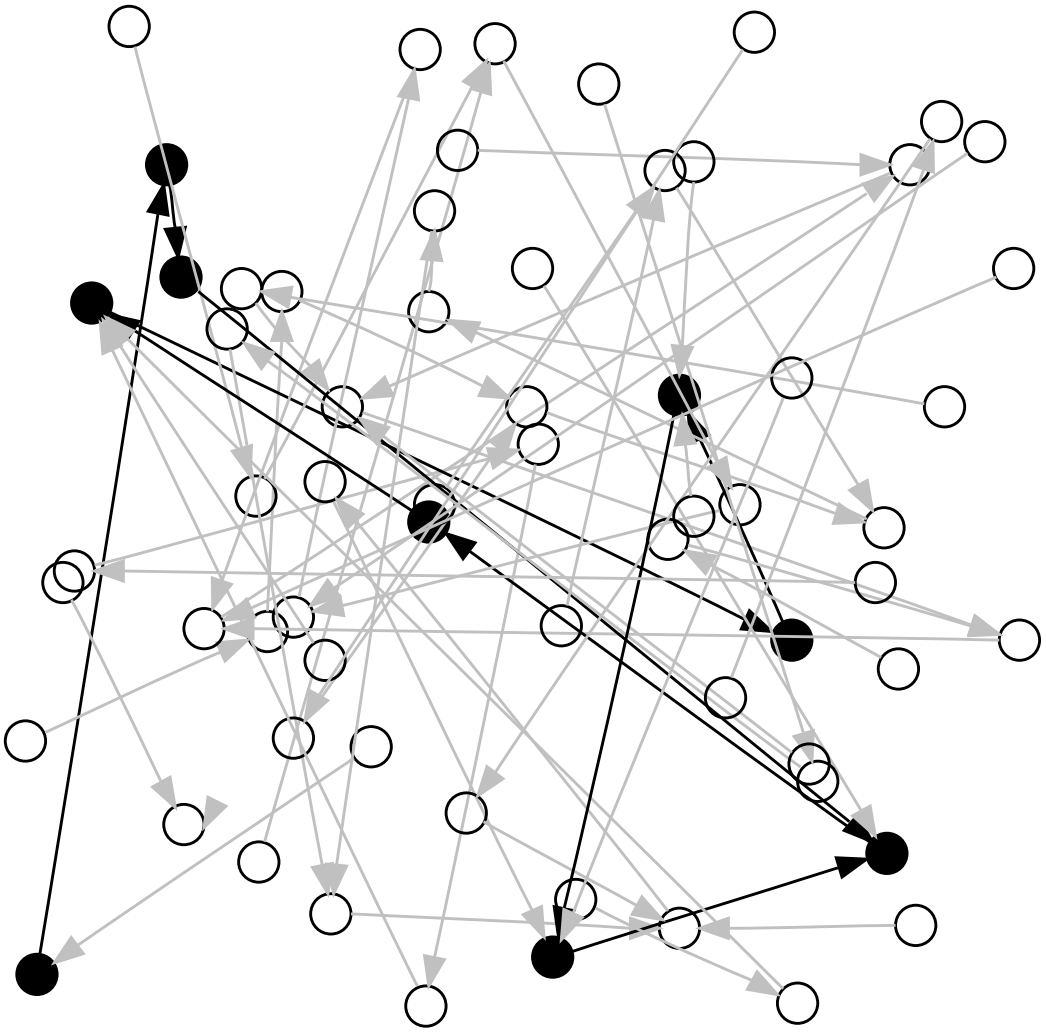


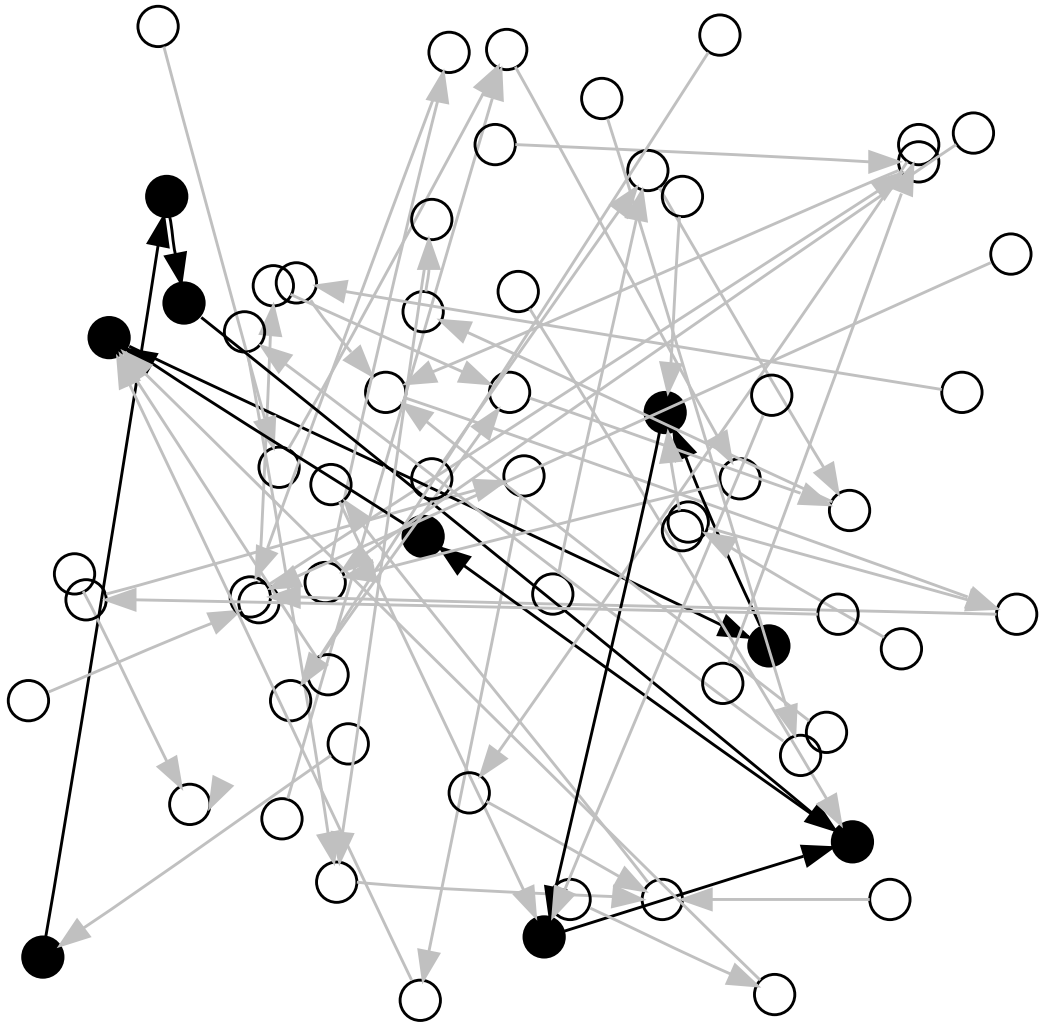


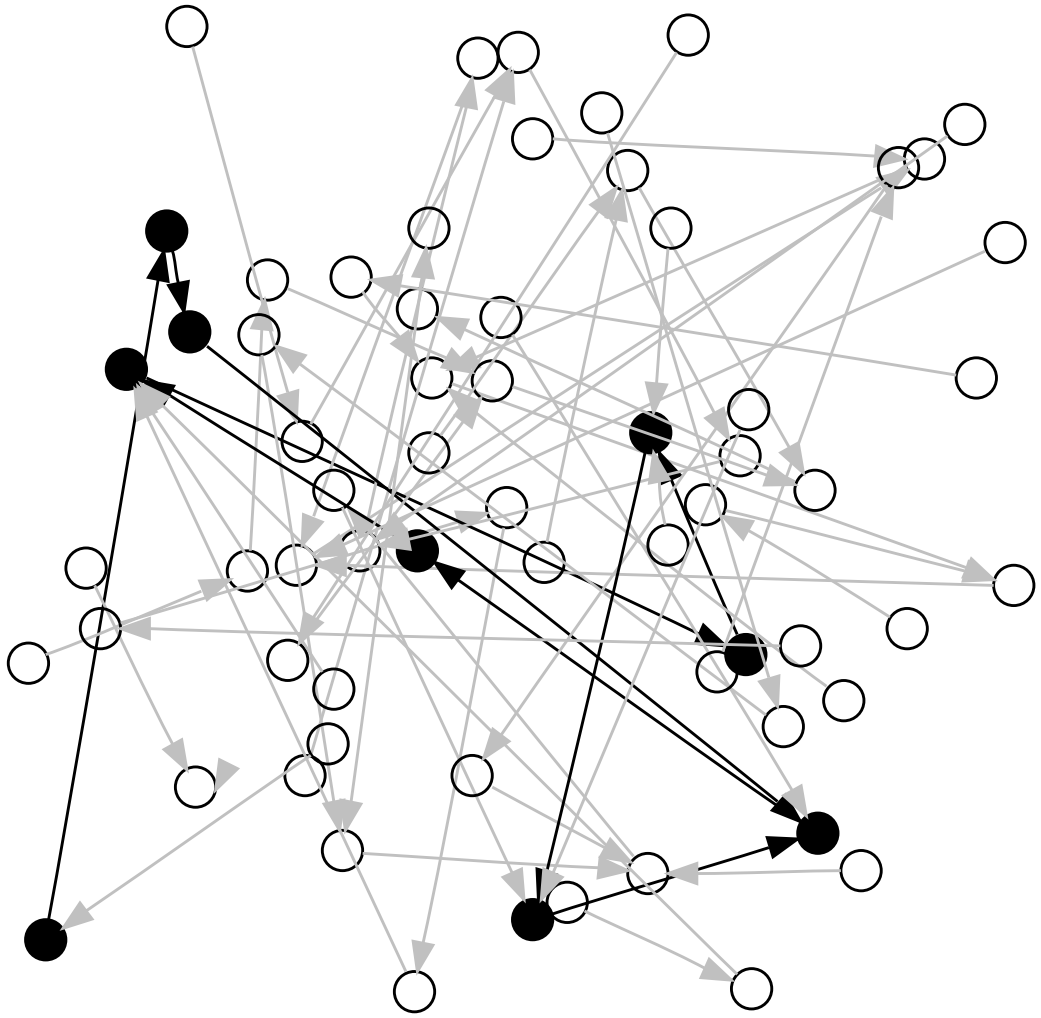


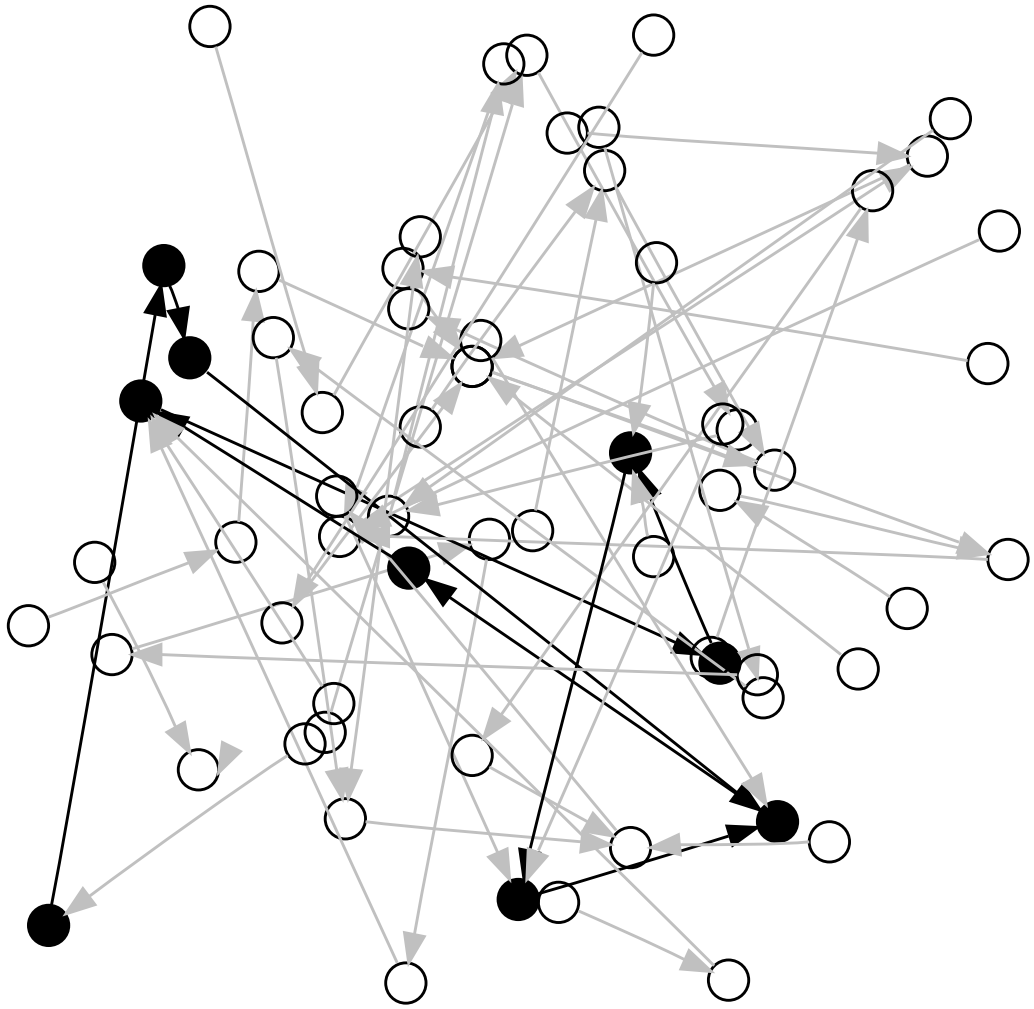


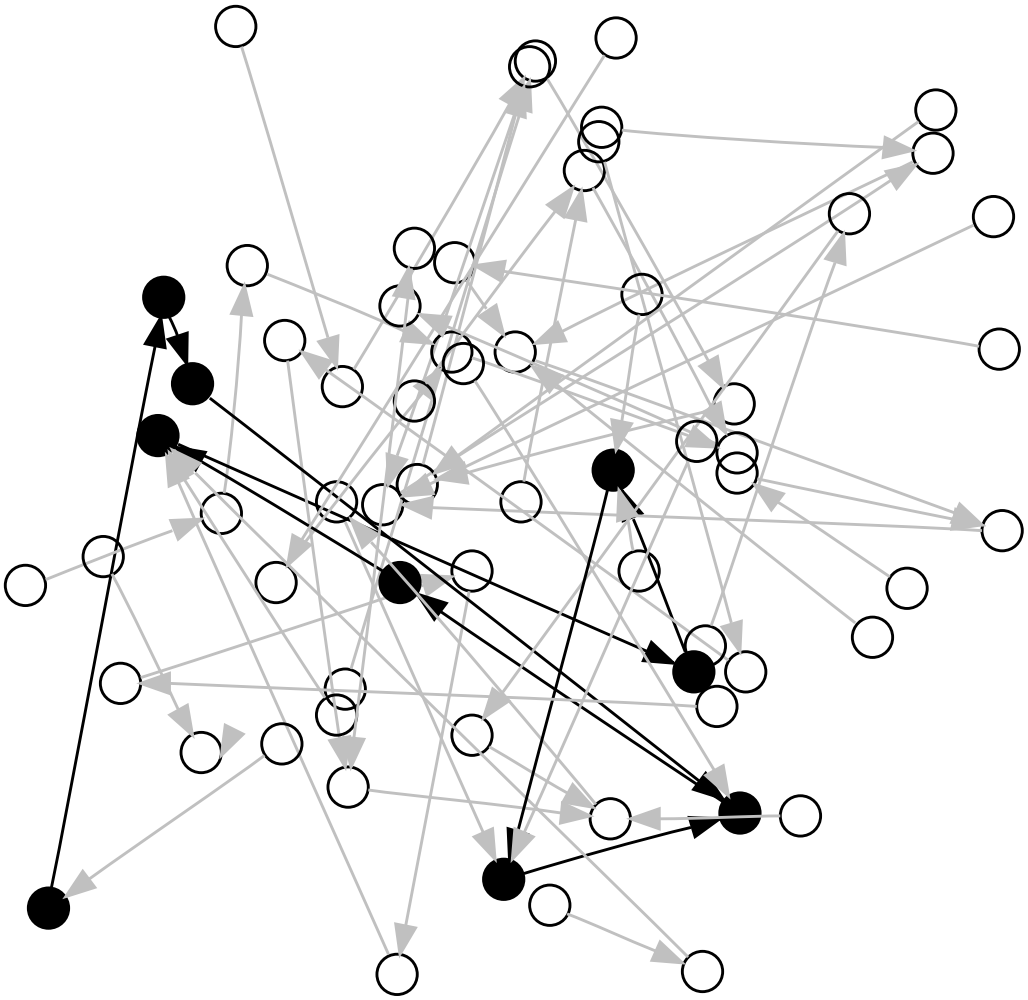


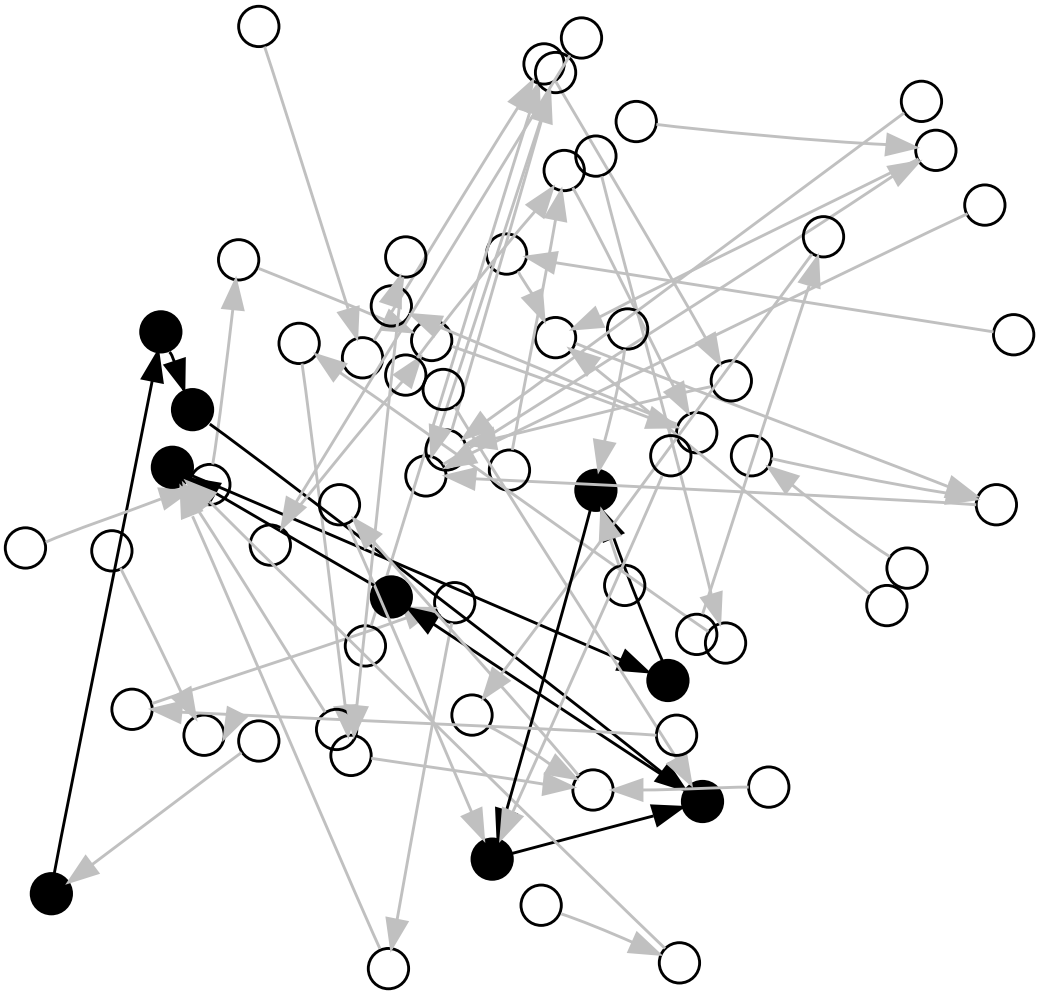


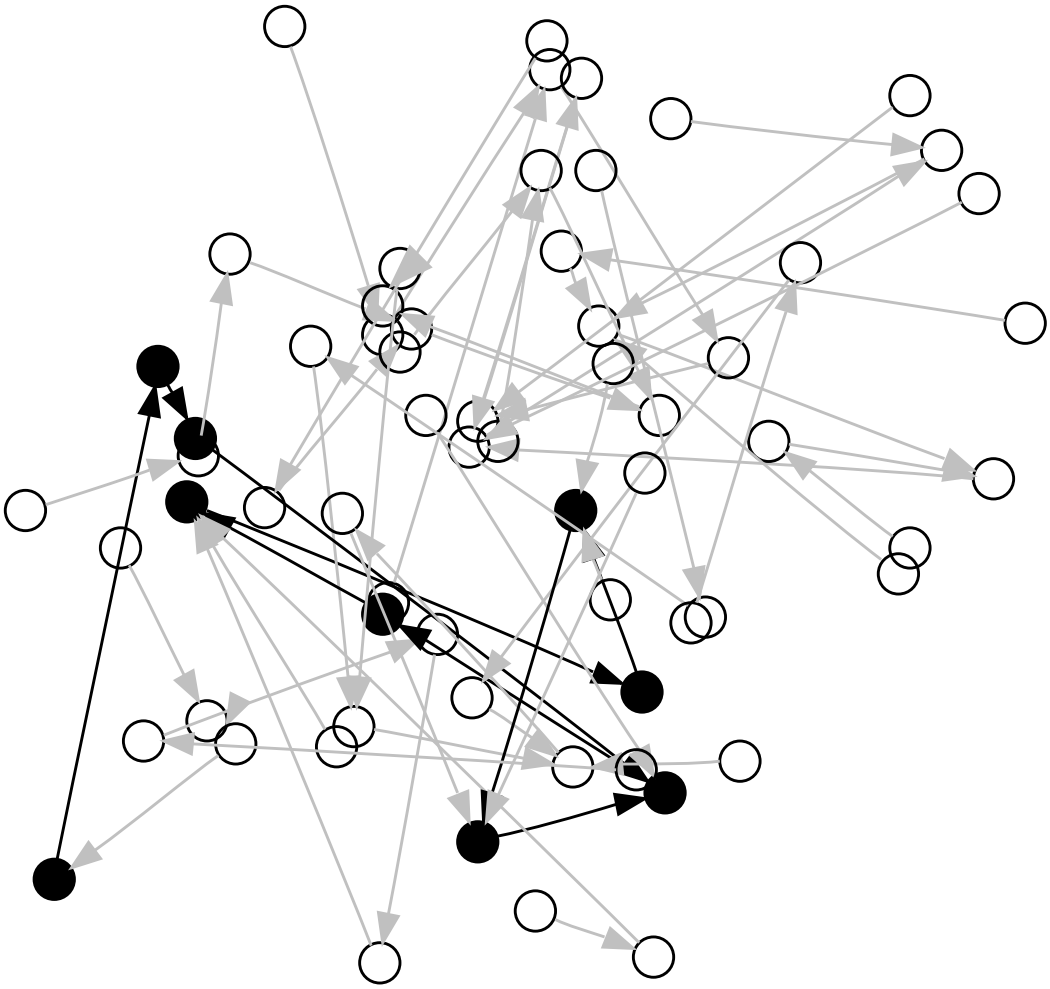


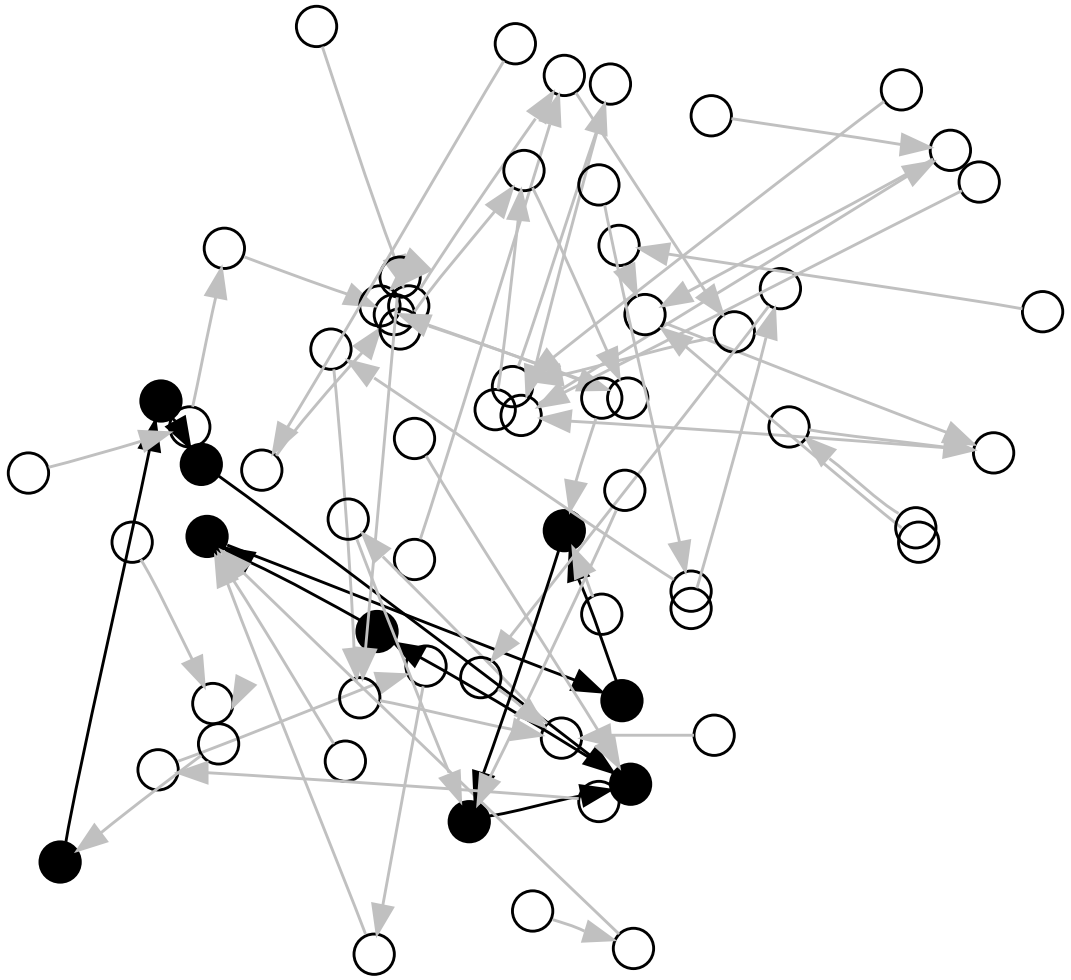


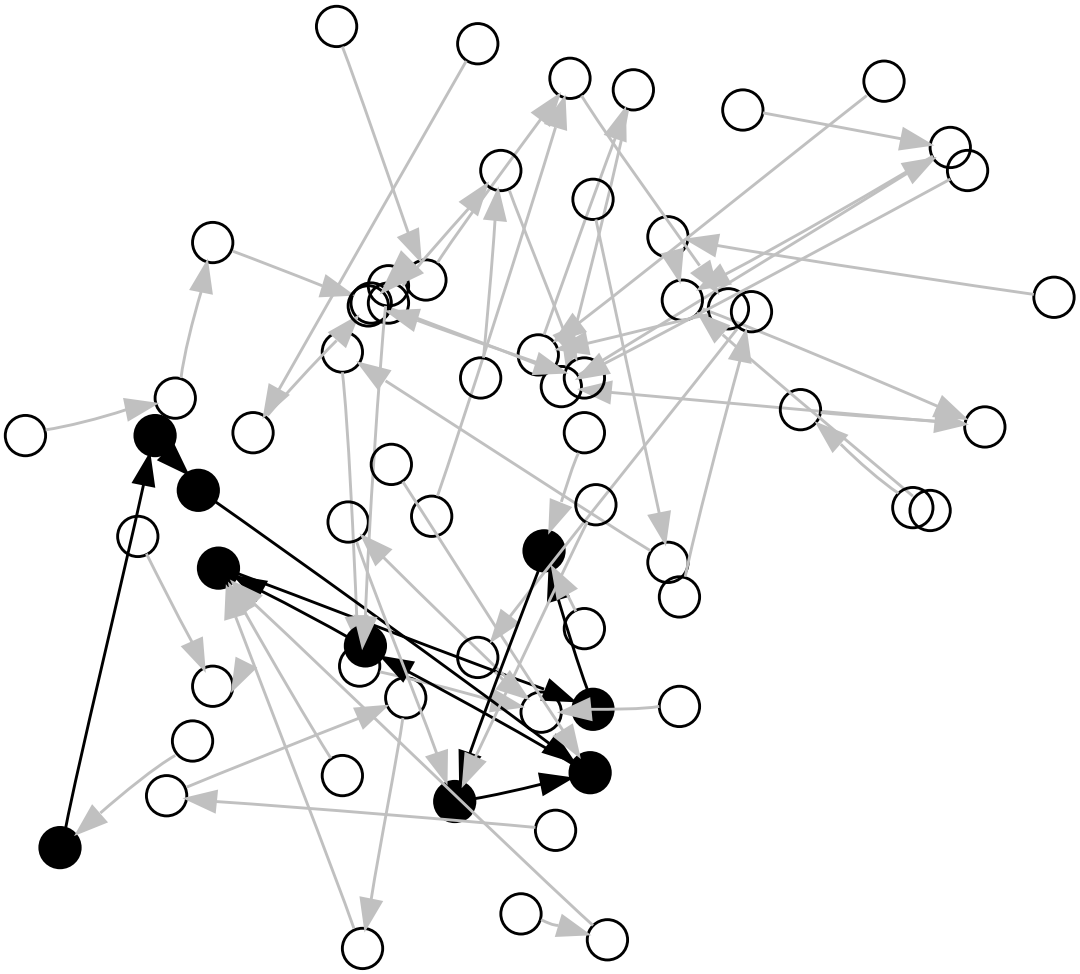


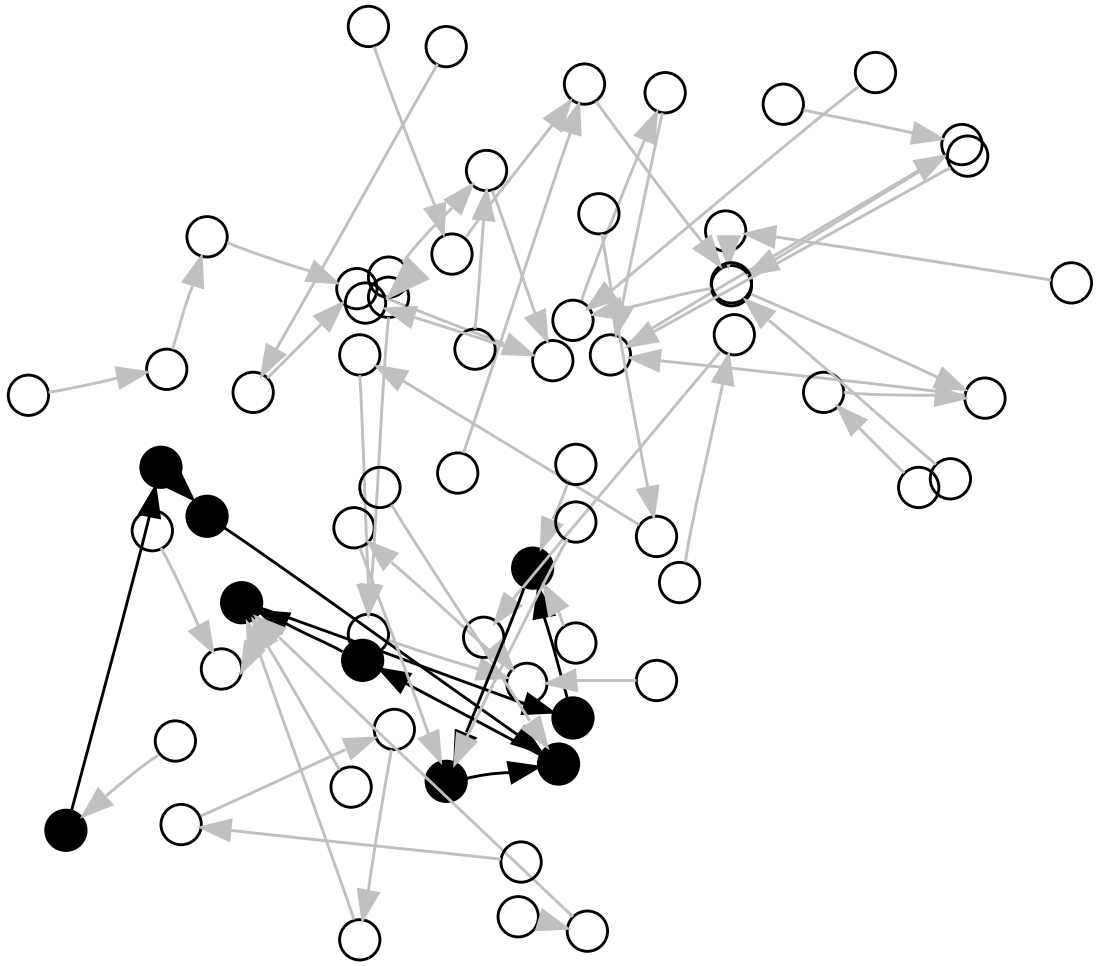


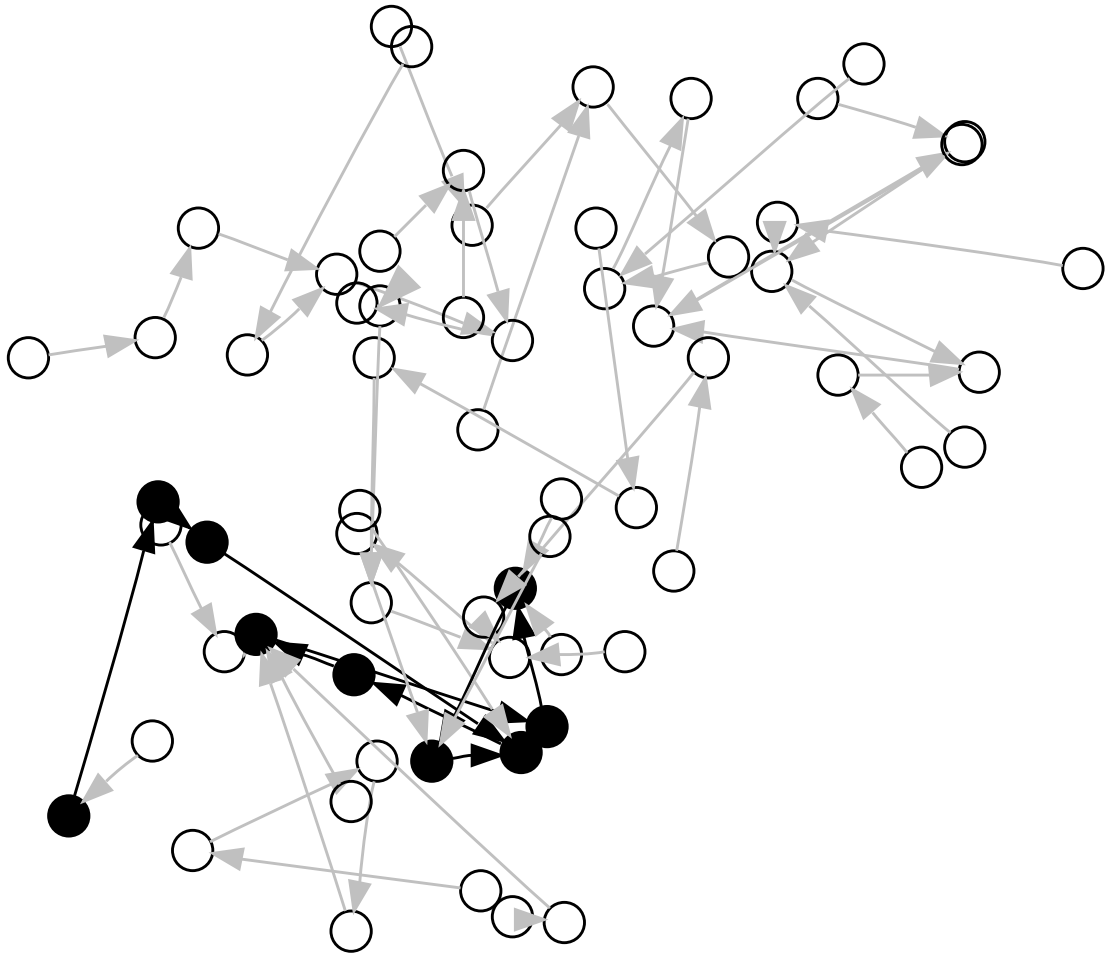


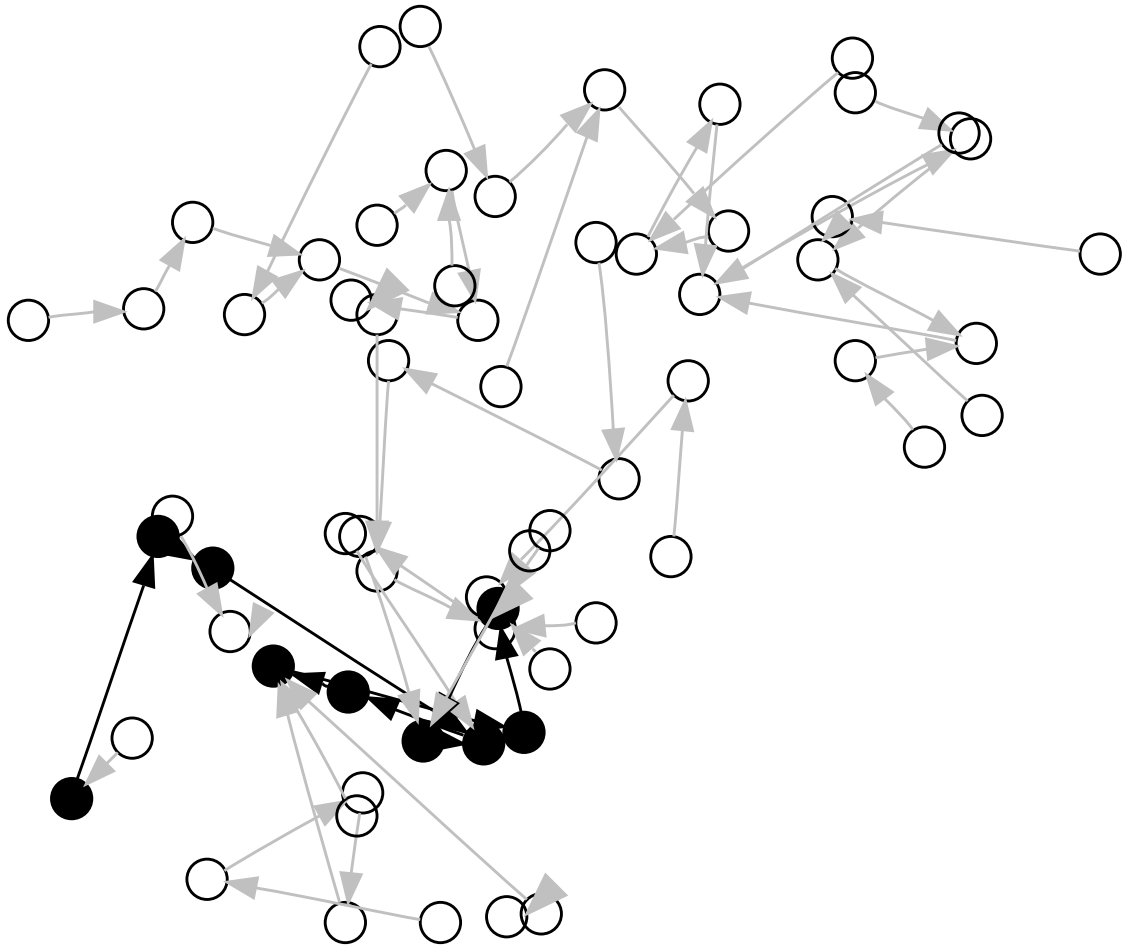


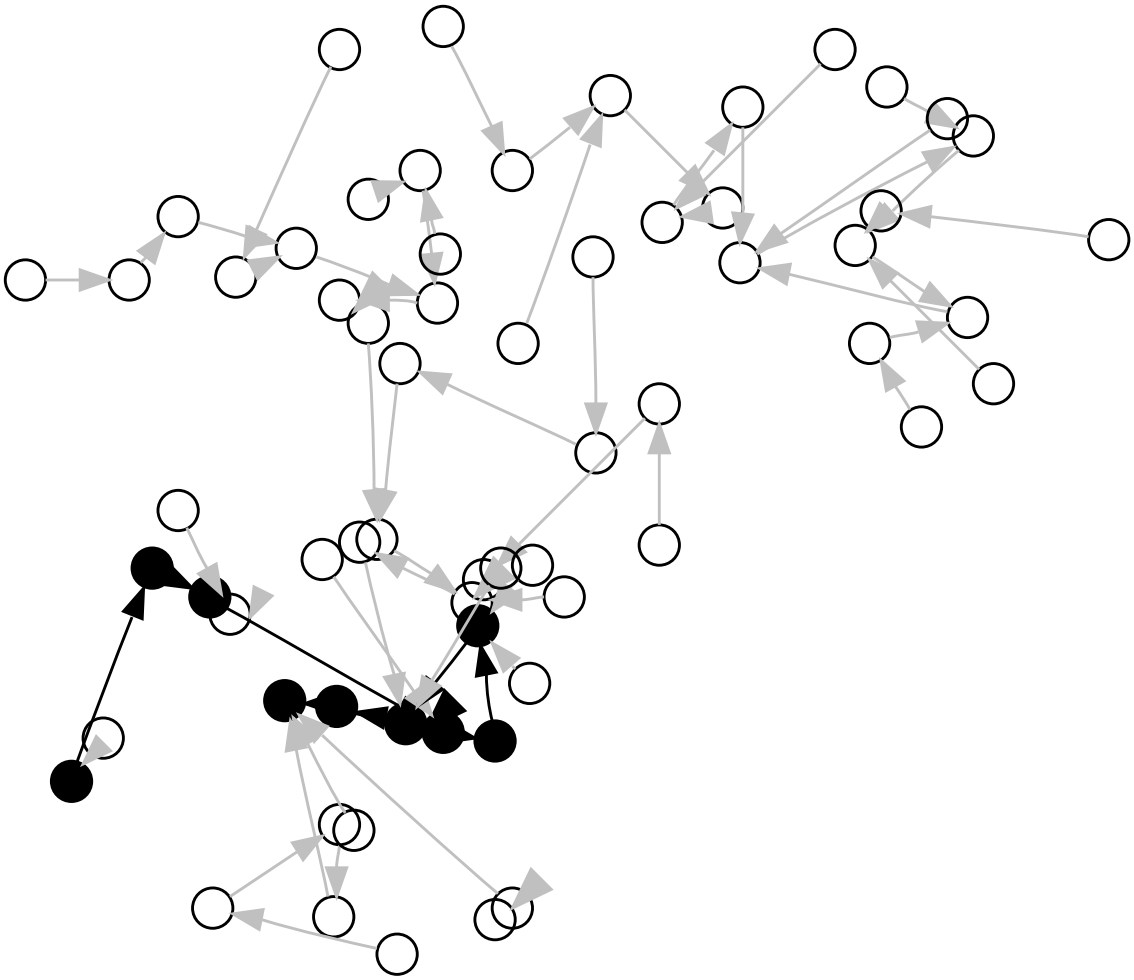


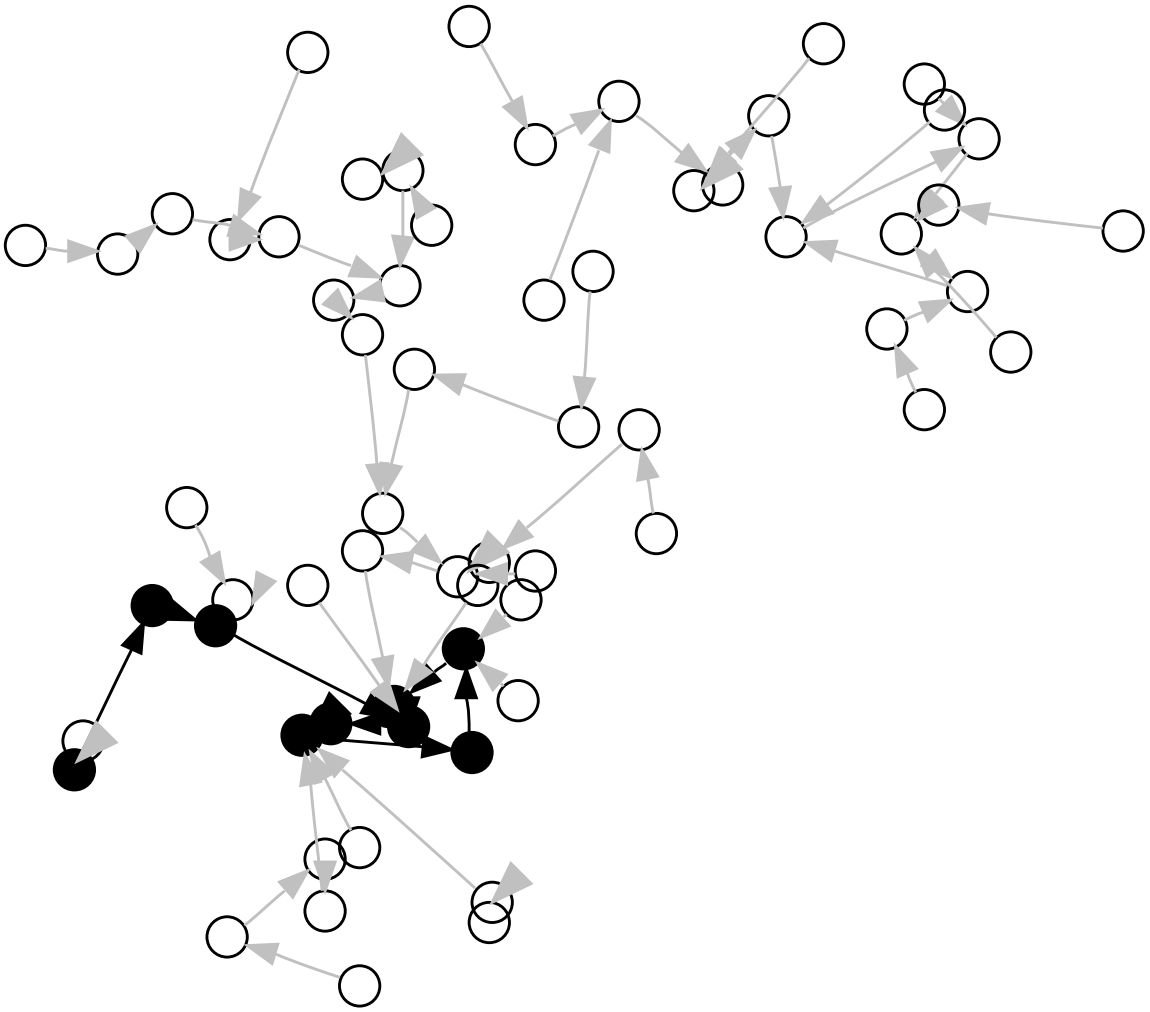


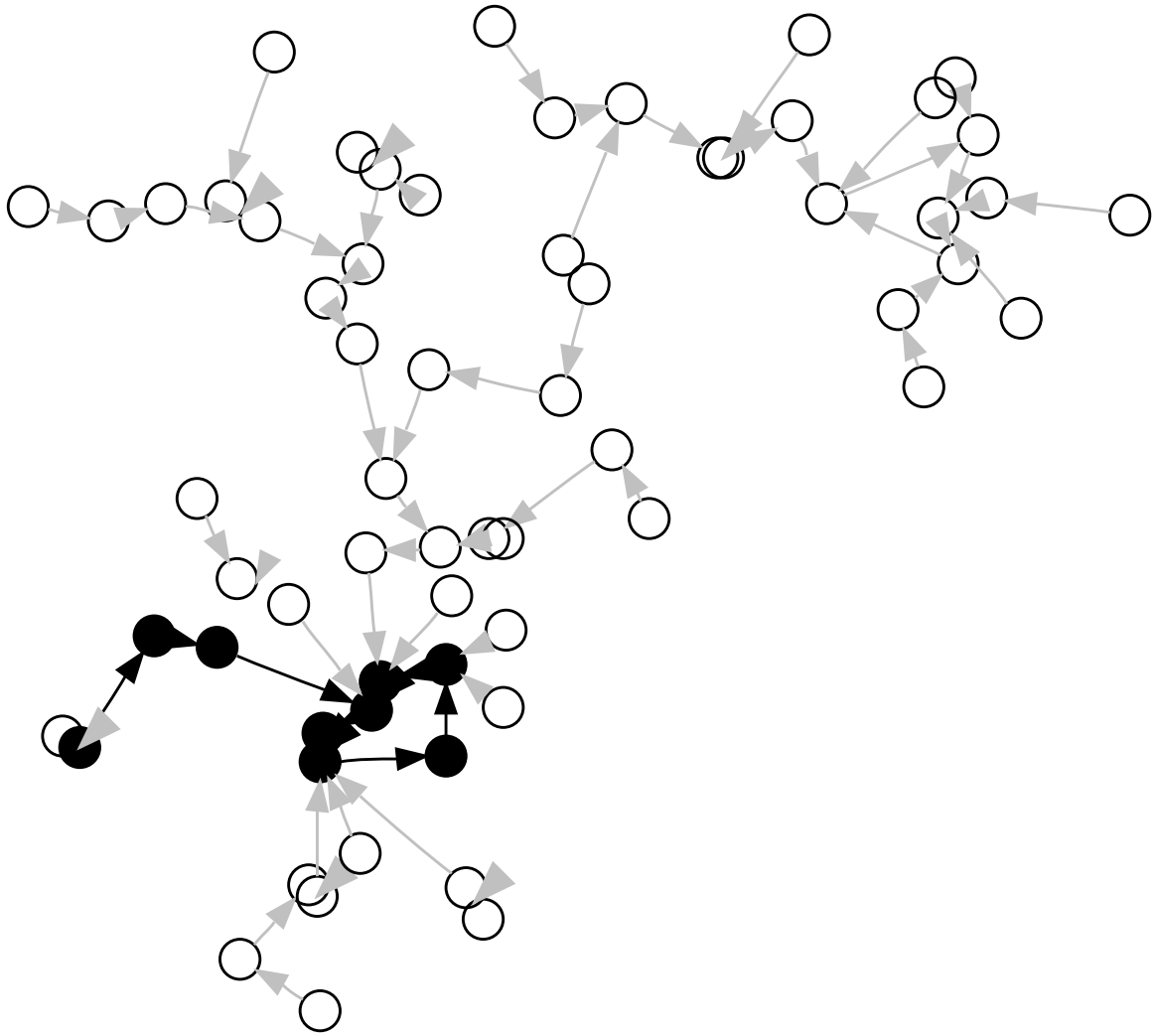


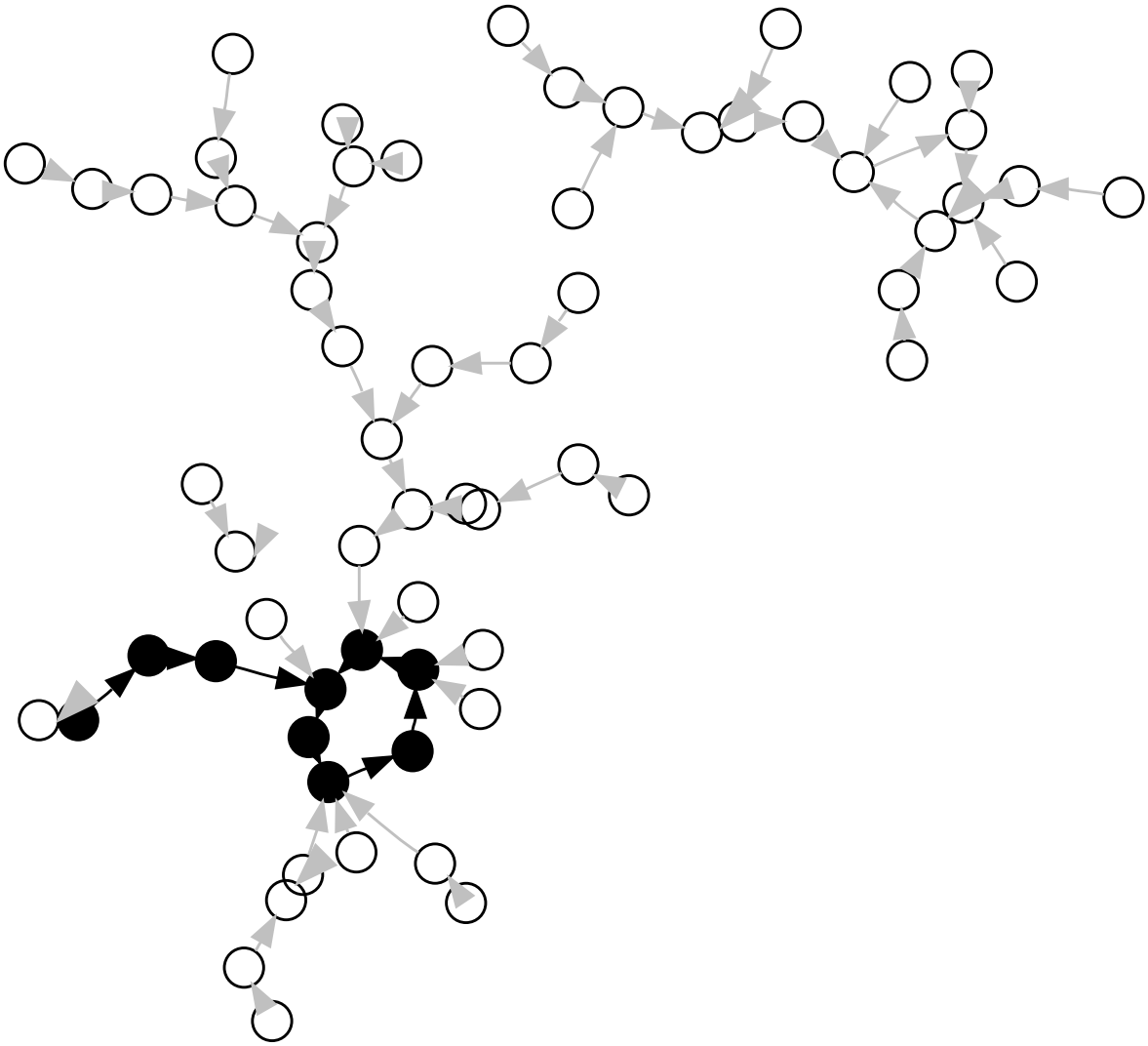


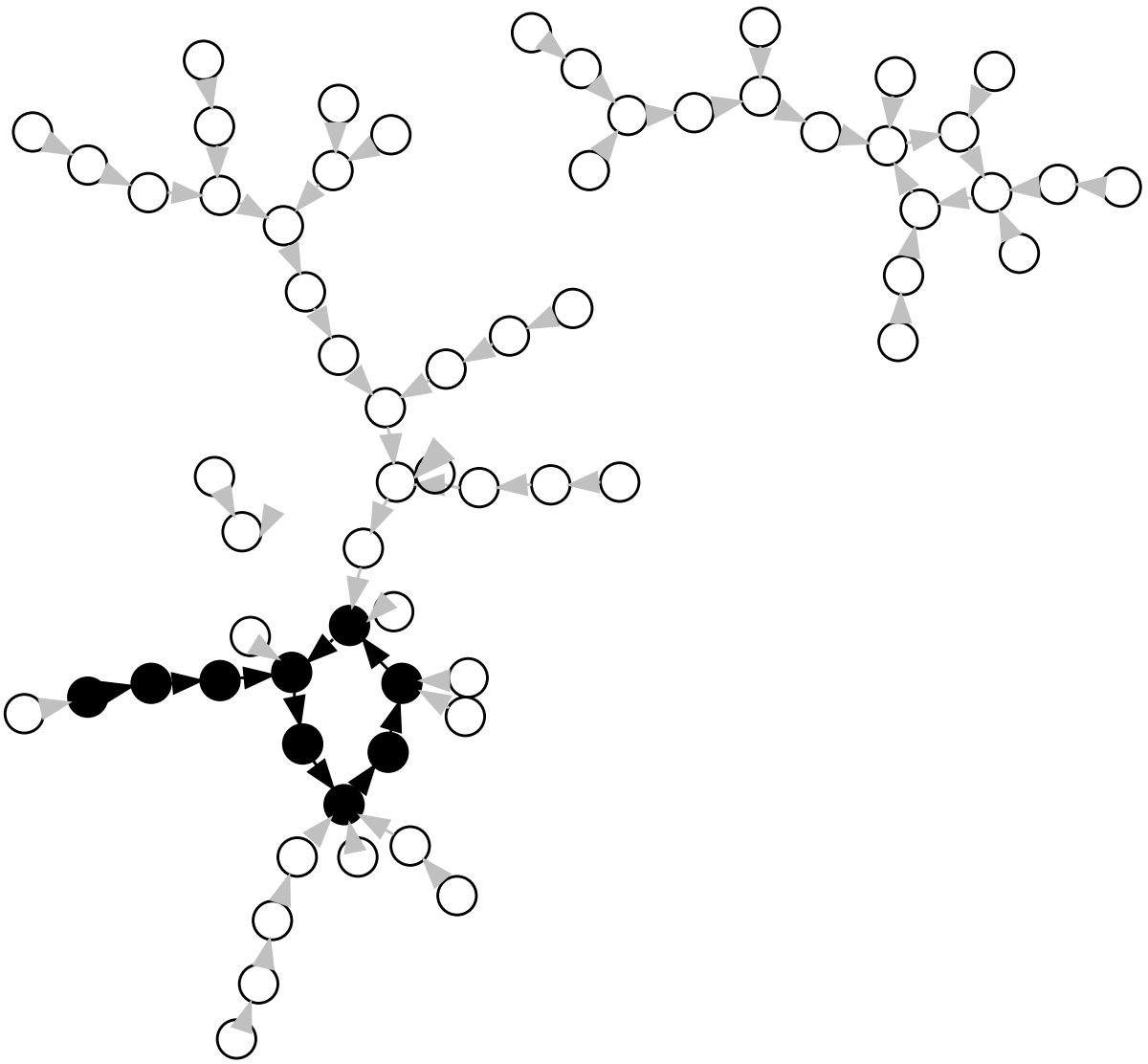












Birthday paradox:
Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

The walk now enters a cycle.

Cycle-finding algorithm
(e.g., Floyd) quickly detects this.

Assume that for each point
we know $a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$
so that $W_i = a_i P + b_i Q$.

Then $W_i = W_j$ means that
$a_i P + b_i Q = a_j P + b_j Q$
so $(b_i - b_j)Q = (a_j - a_i)P$.
If $b_i \neq b_j$ the DLP is solved:
$n = (a_j - a_i)/(b_i - b_j) \bmod \ell$.

Assume that for each point
we know $a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$
so that $W_i = a_i P + b_i Q$.

Then $W_i = W_j$ means that
$a_i P + b_i Q = a_j P + b_j Q$
so $(b_i - b_j)Q = (a_j - a_i)P$.
If $b_i \neq b_j$ the DLP is solved:
$n = (a_j - a_i)/(b_i - b_j) \bmod \ell$.

E.g. $f(W_i) = a(W_i)P + b(W_i)Q$,
starting from some initial
combination $W_0 = a_0 P + b_0 Q$.
If any W_i and W_j collide then
$W_{i+1} = W_{j+1}$, $W_{i+2} = W_{j+2}$,
etc.

If functions $a(W)$ and $b(W)$ are random modulo ℓ, iterations perform a random walk in $\langle P \rangle$.
If a and b are chosen such that $f(W_i) = f(-W_i)$ then the walk is defined on *equivalence classes* under $\pm$.
There are only $\lceil \ell/2 \rceil$ different classes. This reduces the average number of iterations by a factor of almost exactly $\sqrt{2}$.

In general, Pollard's rho method can be combined with any easily computed group automorphism of small order.

Parallel collision search

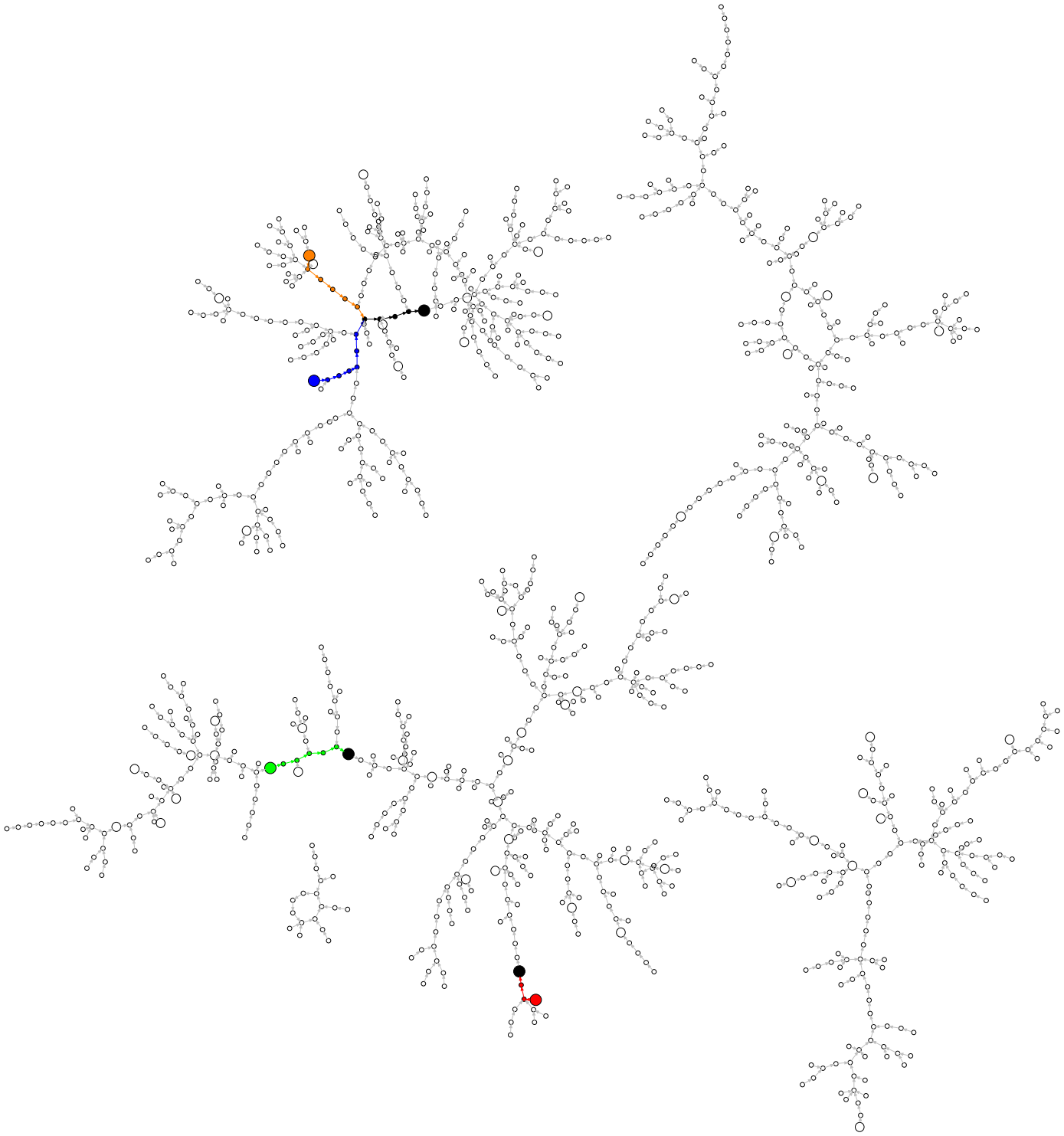
Running Pollard's rho method on N computers gives speedup of $\approx \sqrt{N}$ from increased likelihood of finding collision.

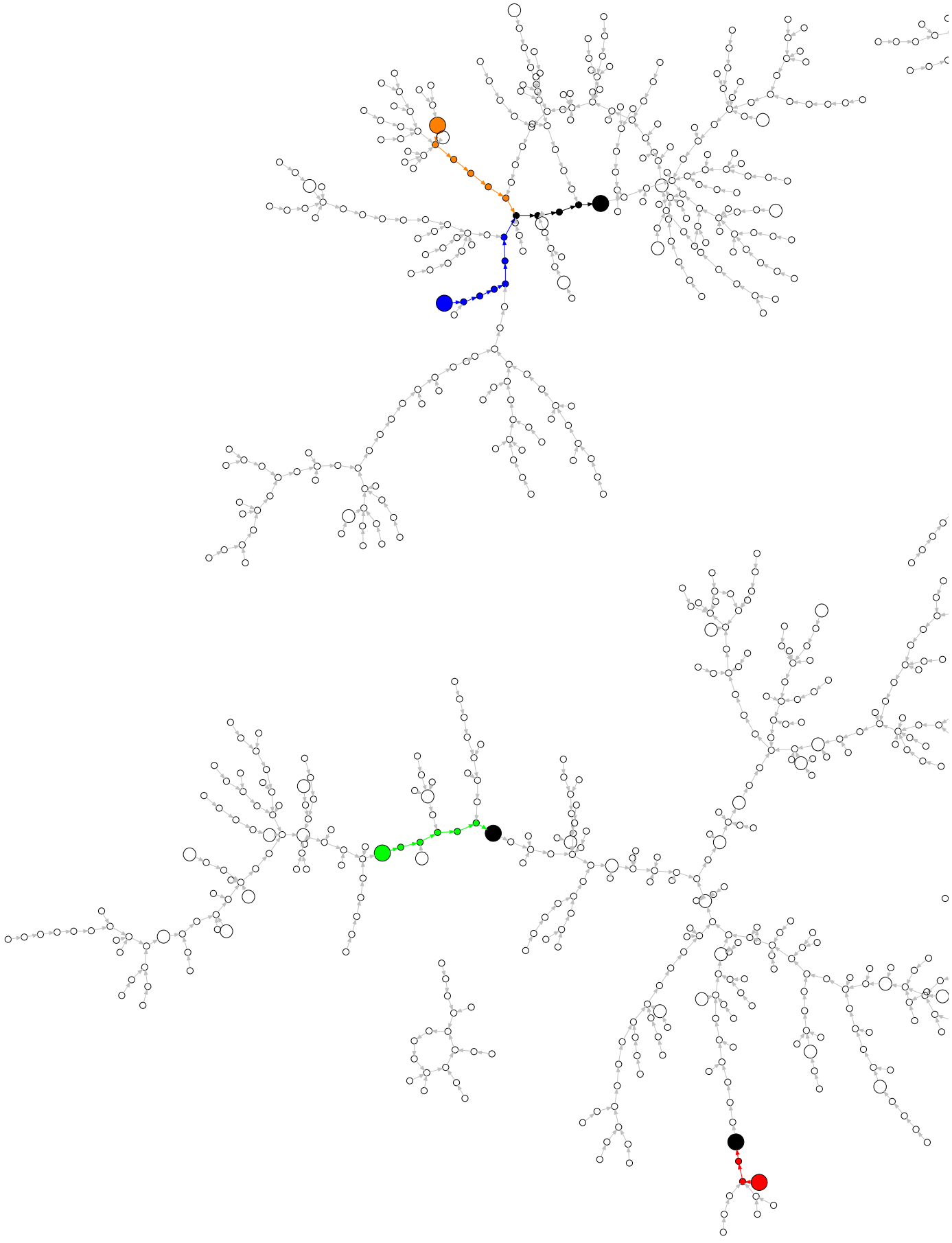
Want better way to spread computation across clients. Want to find collisions between walks on *different* machines, without frequent synchronization!

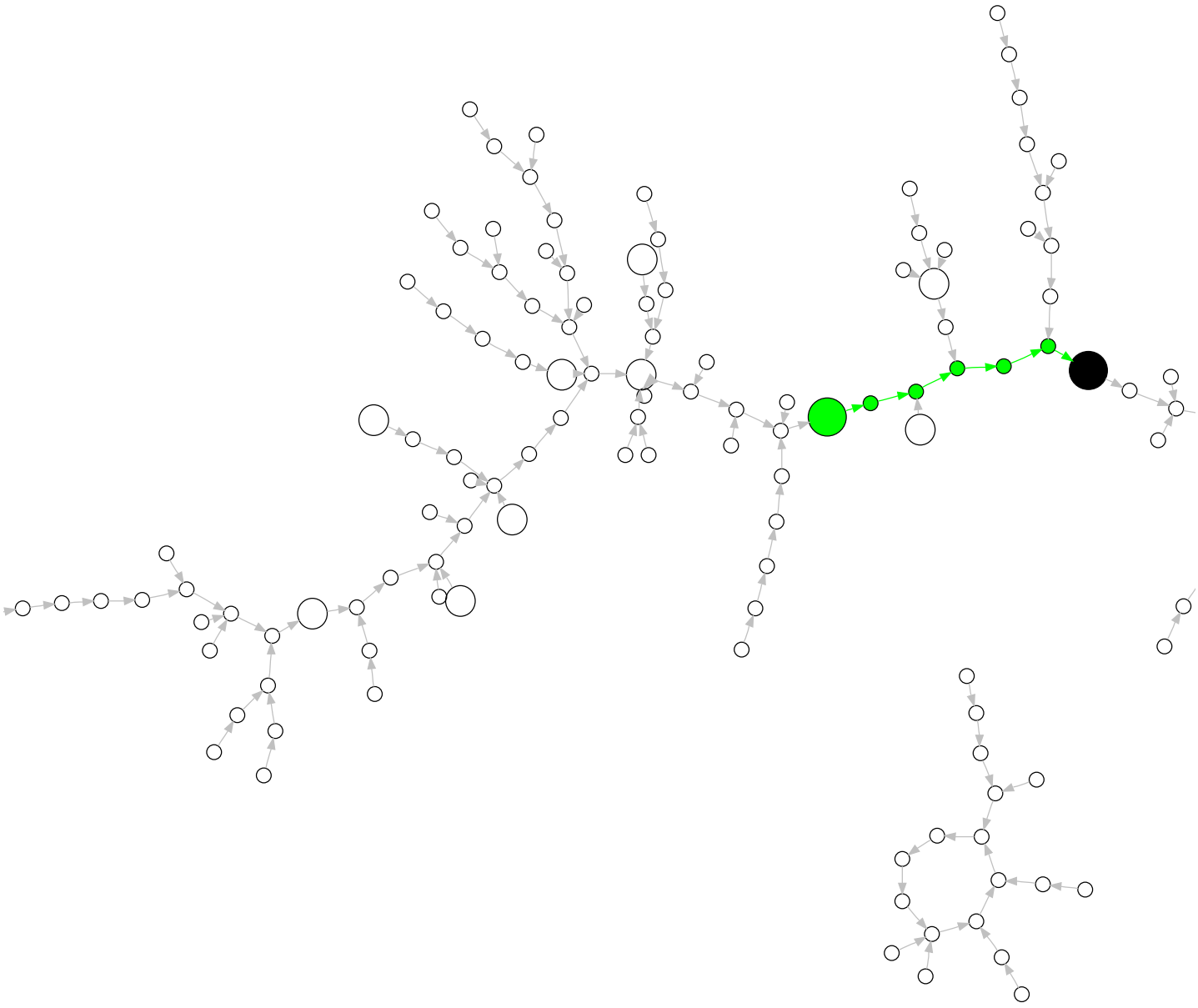
Better method due to van Oorschot and Wiener (1999). Declare some subset of $\langle P \rangle$ to be *distinguished points*.

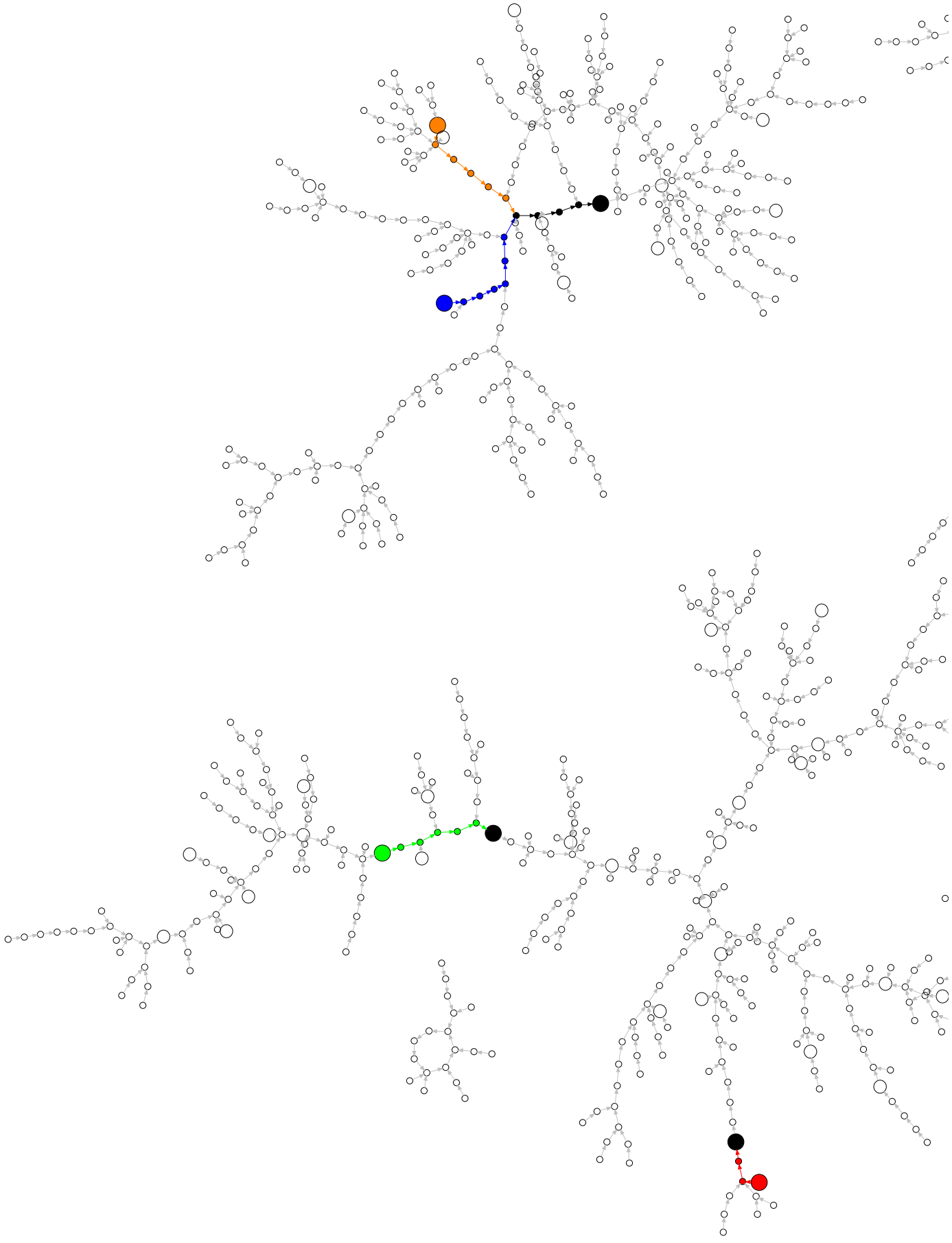
Parallel rho: Perform many walks
with different starting points
but same update function f.
If two different walks
find the same point then
their subsequent steps will match.

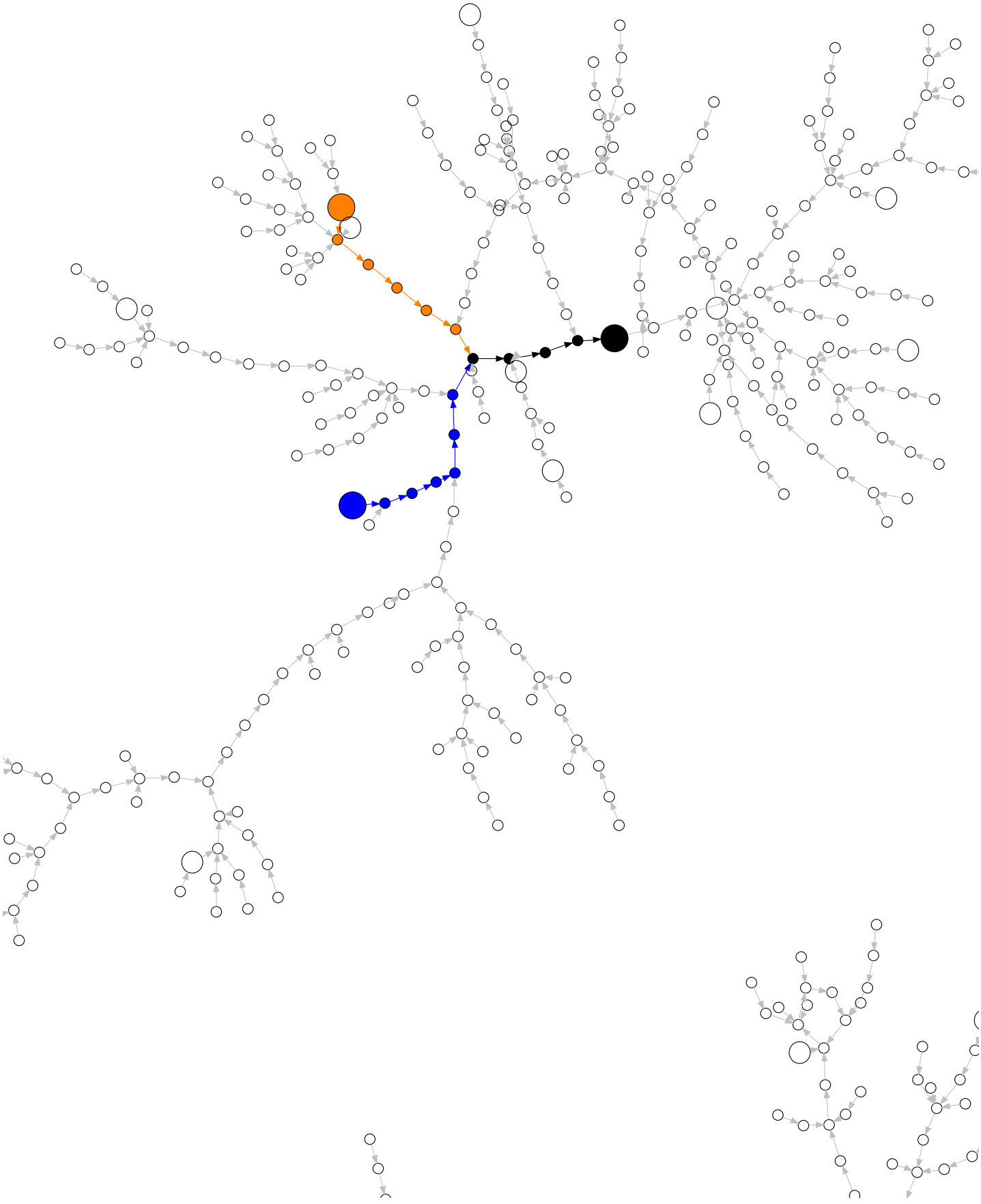
Terminate each walk once it hits
a distinguished point and report
the point along with a_i and b_i to
server.
Server receives, stores, and sorts
all distinguished points.
Two walks reaching same
distinguished point give collision.
This collision solves the DLP.











Attacker chooses frequency and definition of distinguished points.

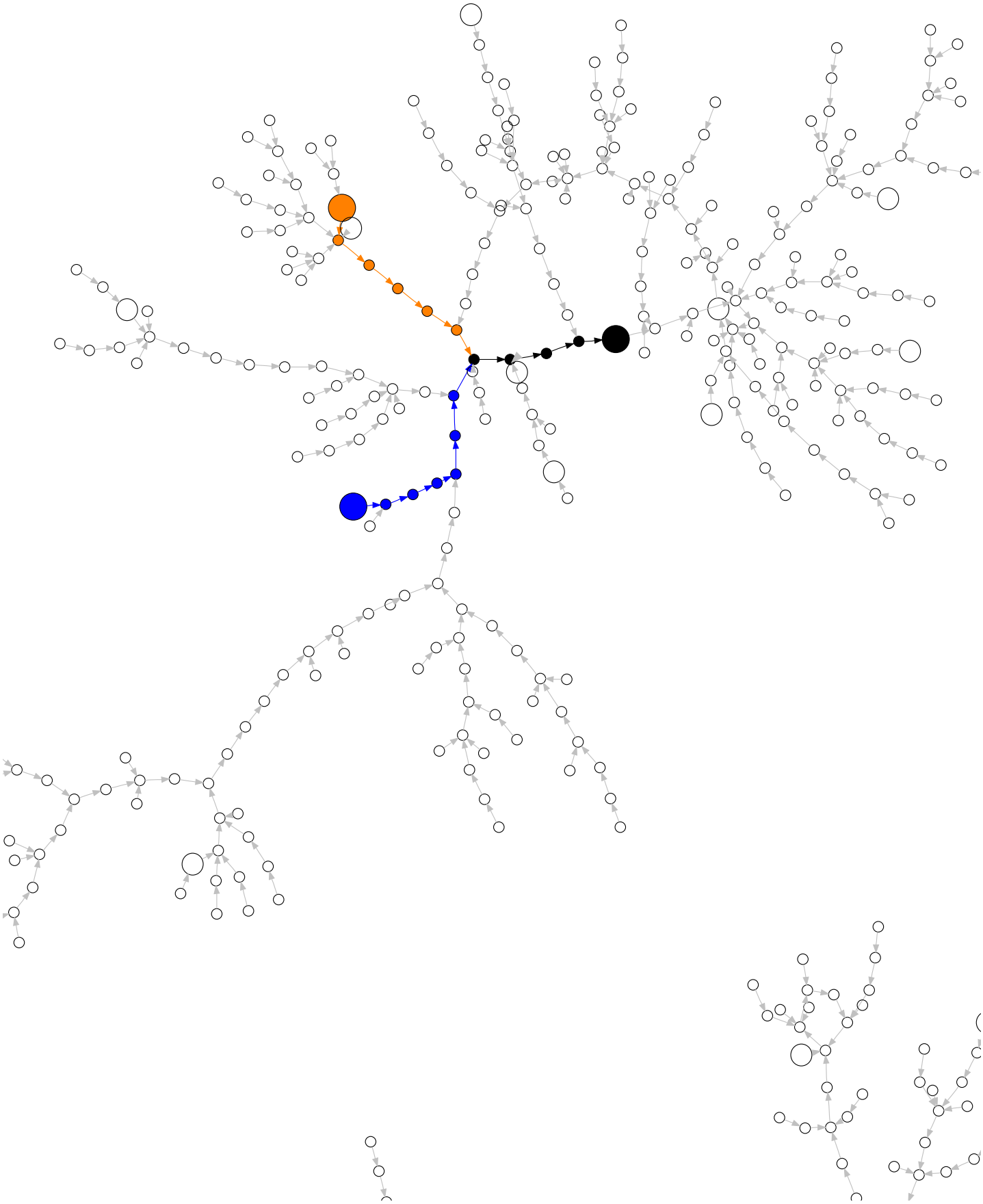
Tradeoffs are possible:

If distinguished points are rare, a small number of very long walks will be performed. This reduces the number of distinguished points sent to the server but increases the delay before a collision is recognized.

If distinguished points are frequent, many shorter walks will be performed.

In any case do not wait for cycle.

Total # of iterations unchanged.



Additive walks

Generic rho method requires two scalar multiplications for each iteration.
Could replace by double-scalar multiplication; could further merge the 2-scalar multiplications across several parallel iterations.

Additive walks

Generic rho method requires two scalar multiplications for each iteration.
Could replace by double-scalar multiplication; could further merge the 2-scalar multiplications across several parallel iterations.

More efficient: use *additive walk*:
Start with $W_0 = a_0 P + b_0 Q$ & put $f(W_i) = W_i + c_j P + d_j Q$ where $j = h(W_i)$.

Pollard's initial proposal:

Use $x(W_i) \bmod 3$ as h

and update:

$$W_{i+1} = \begin{cases} W_i + P \text{ for } x(W_i) \bmod 3 = 0 \\ 2W_i \quad \text{for } x(W_i) \bmod 3 = 1 \\ W_i + Q \text{ for } x(W_i) \bmod 3 = 2 \end{cases}$$

Easy to update a_i and b_i.

$$(a_{i+1}, b_{i+1}) = \begin{cases} (a_i + 1, b_i) \text{ for } x(W_i) \bmod 3 = 0 \\ (2a_i, 2b_i) \quad \text{for } x(W_i) \bmod 3 = 1 \\ (a_i, b_i + 1) \text{ for } x(W_i) \bmod 3 = 2 \end{cases}$$

Additive walk requires only one addition per iteration.
h maps from $\langle P \rangle$ to $\{0, 1, \ldots, r-1\}$, and $R_j = c_j P + d_j Q$ are precomputed for each $j \in \{0, 1, \ldots, r-1\}$.

Easy coefficient update:
$W_i = a_i P + b_i Q$,
where a_i and b_i are defined recursively as follows:
$a_{i+1} = a_i + c_{h(W_i)}$ and
$b_{i+1} = b_i + d_{h(W_i)}$.

Additive walks have disadvantages:
The walks are noticeably nonrandom; this means they need more iterations than the generic rho method to find a collision.

This effect disappears as r grows, but but then the precomputed table $R_0, \ldots, R_{r-1}$ does not fit into fast memory. This depends on the platform, e.g. trouble for GPUs.
There is more trouble with adding walks,

e.g., in combination with negation.

Randomness of adding walks

Let $h(W) = i$ with probability p_i.

Fix a point T, and let W and W' be two independent uniform random points.

Let $W \neq W'$ both map to T.

This event occurs if

Randomness of adding walks

Let $h(W) = i$ with probability p_i.

Fix a point T, and let W and W' be two independent uniform random points.

Let $W \neq W'$ both map to T.

This event occurs if simultaneously for $i \neq j$:
$T = W + R_i = W' + R_j$;
$h(W) = i$; $h(W') = j$.

These conditions have probability $1/\ell^2$, p_i, and p_j respectively.

Summing over all (i, j) gives the overall probability $\left(\sum_{i \neq j} p_i p_j\right) / \ell^2 = \left(\sum_{i,j} p_i p_j - \sum_i p_i^2\right) / \ell^2 = \left(1 - \sum_i p_i^2\right) / \ell^2$.

This means that the probability of an immediate collision from W and W' is $\left(1 - \sum_i p_i^2\right) / \ell$, where we added over the ℓ choices of T. In the simple case that all the p_i are $1/r$, the difference from the optimal $\sqrt{\pi \ell / 2}$ iterations is a factor of $1/\sqrt{1 - 1/r} \approx 1 + 1/(2r)$.

Various heuristics leading to standard $\sqrt{1-1/r}$ formula in different ways:

1981 Brent–Pollard;

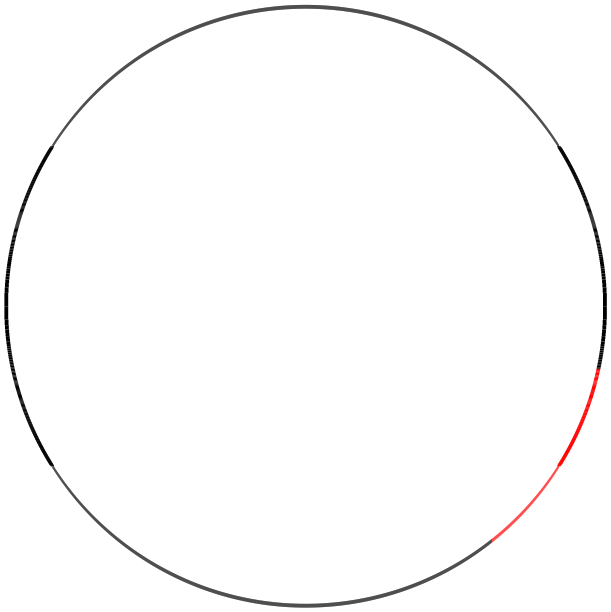
2001 Teske;

2009 ECC2K-130 paper, eprint 2009/541.

Various heuristics leading to standard $\sqrt{1-1/r}$ formula in different ways:
1981 Brent–Pollard;
2001 Teske;
2009 ECC2K-130 paper, eprint 2009/541.

2010 Bernstein–Lange:
Standard formula is wrong!
There is a further slowdown from higher-order anti-collisions:
e.g. $W + R_i + R_k \neq W' + R_j + R_l$ if $R_i + R_k = R_j + R_l$.
$\approx 1\%$ slowdown for ECC2K-130.

DLs in intervals



Want to use knowledge
that DL is in a
small interval $[a, b]$,
much smaller than ℓ.

We can use this in baby-step giant-step algorithm.

How to use this in a memory-less algorithm?

Standard interval method:
Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.

Standard interval method:

Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.

Real kangaroos sleep

Standard interval method:
Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.

Real kangaroos sleep

(at least outside Australia).

Kangaroo method

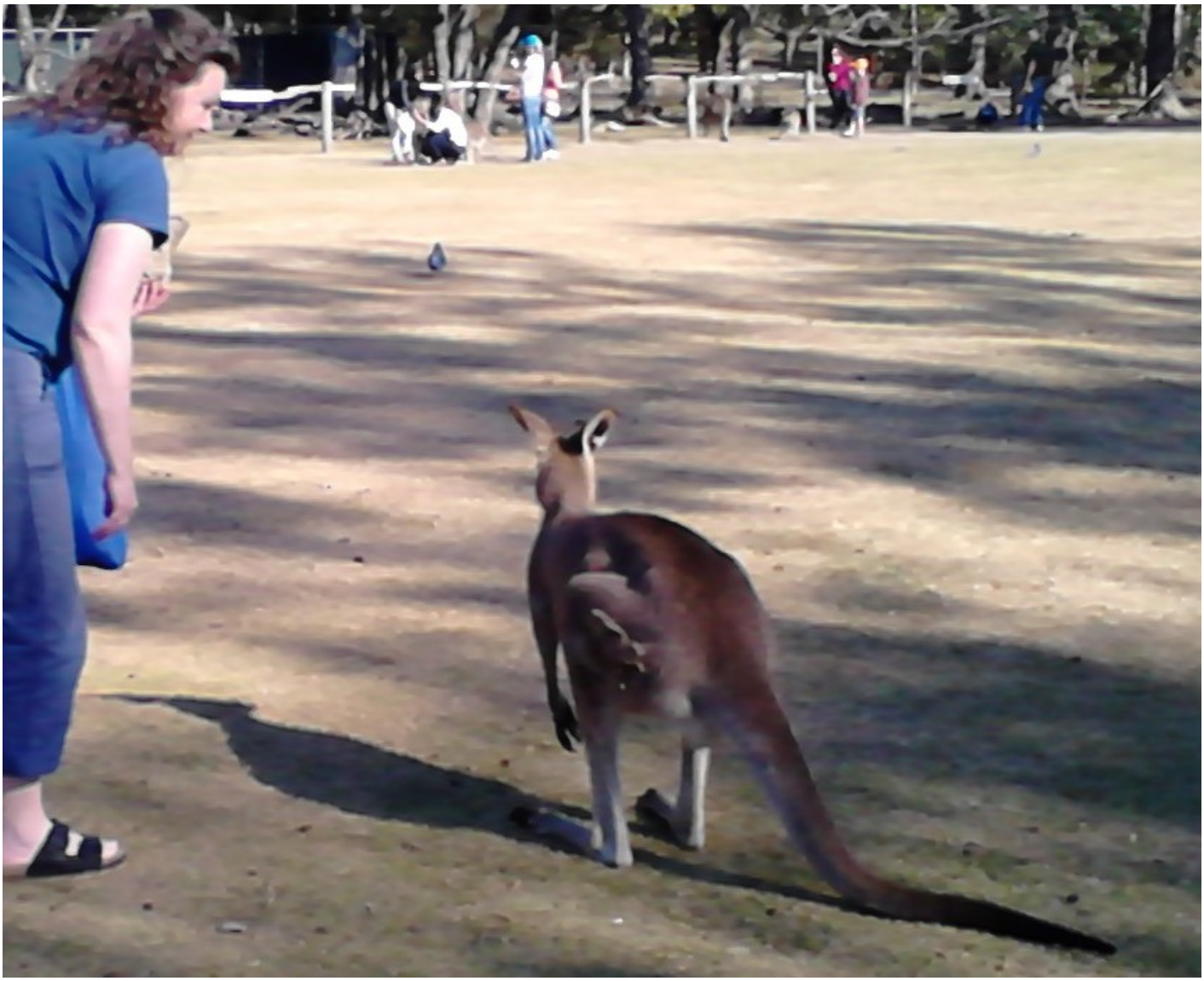
in Australia

Main actor:

The tame kangaroo

starts at a known
multiple of P, e.g. bP.

The tame kangaroo jumps.



Jumps are determined
by current position.

The tame kangaroo jumps.

Jumps are determined by current position.

Average jump distance is $\sqrt{b-a}$.

The tame kangaroo jumps.

Jumps are determined by current position.

Average jump distance is $\sqrt{b-a}$.

The tame kangaroo jumps.

Jumps are determined

by current position.

Average jump distance

is $\sqrt{b-a}$.

The tame kangaroo stops

after a fixed number of jumps (about $\sqrt{b-a}$ many).

The tame kangaroo installs a trap and waits.

The wild kangaroo

starts at point Q.

Follows the same instructions for jumps.

But we don't know where
the starting point Q is.
Know $Q = nP$ with $n \in [a, b]$.

Hope that the paths of the tame
and wild kangaroo intersect.

Similar to the rho method the
kangaroos will hop on the same
path from that point onwards.

Eventually the wild kangaroo falls
into the trap.
(Or disappears in the distance if
paths have not intersected.
Start a fresh one
from $Q + P, Q + 2P, \ldots$)

Same story in math

Kangaroo = sequence $X_i \in \langle P \rangle$.

Starting point $X_0 = s_0 P$.

Distance $d_0 = 0$.

Step set: $S = \{s_1 P, \ldots, s_L P\}$,

with s_i on average

$s = \beta\sqrt{b-a}$.

Hash function

$H : \langle P \rangle \to \{1, 2, \ldots, L\}$.

Update function

$$d_{i+1} = d_i + s_{H(X_i)}, \quad i = 0, 1, 2, \ldots,$$

$$X_{i+1} = X_i + s_{H(X_i)} P, \quad i = 0, 1, 2, \ldots$$

Tame kangaroo starts at

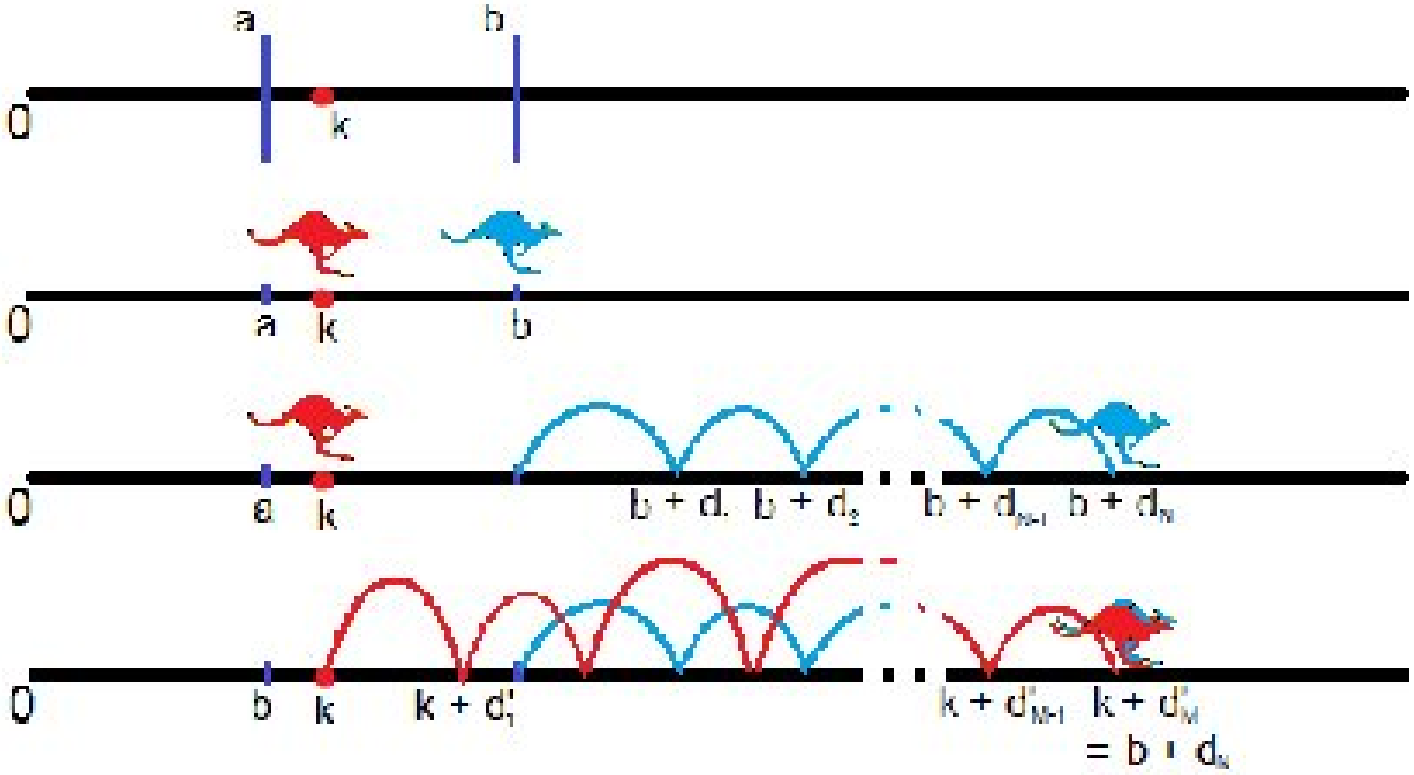
$X_0 = bP$,

wild kangaroo starts at

$X_0' = Q = nP$.

Trap: distance d_N,

endpoint $X_N = (b + d_N)P$.



Picture credit:

Christine van Vredendaal.

Parallel kangaroo method

Use an entire herd

of tame kangaroos,

all starting

around $((b-a)/2)P$ ...

. . . and define certain spots as distinguished points

Also start a herd of
wild kangaroos around Q.
Hope that one wild and
one tame kangaroo
meet at one distinguished point.