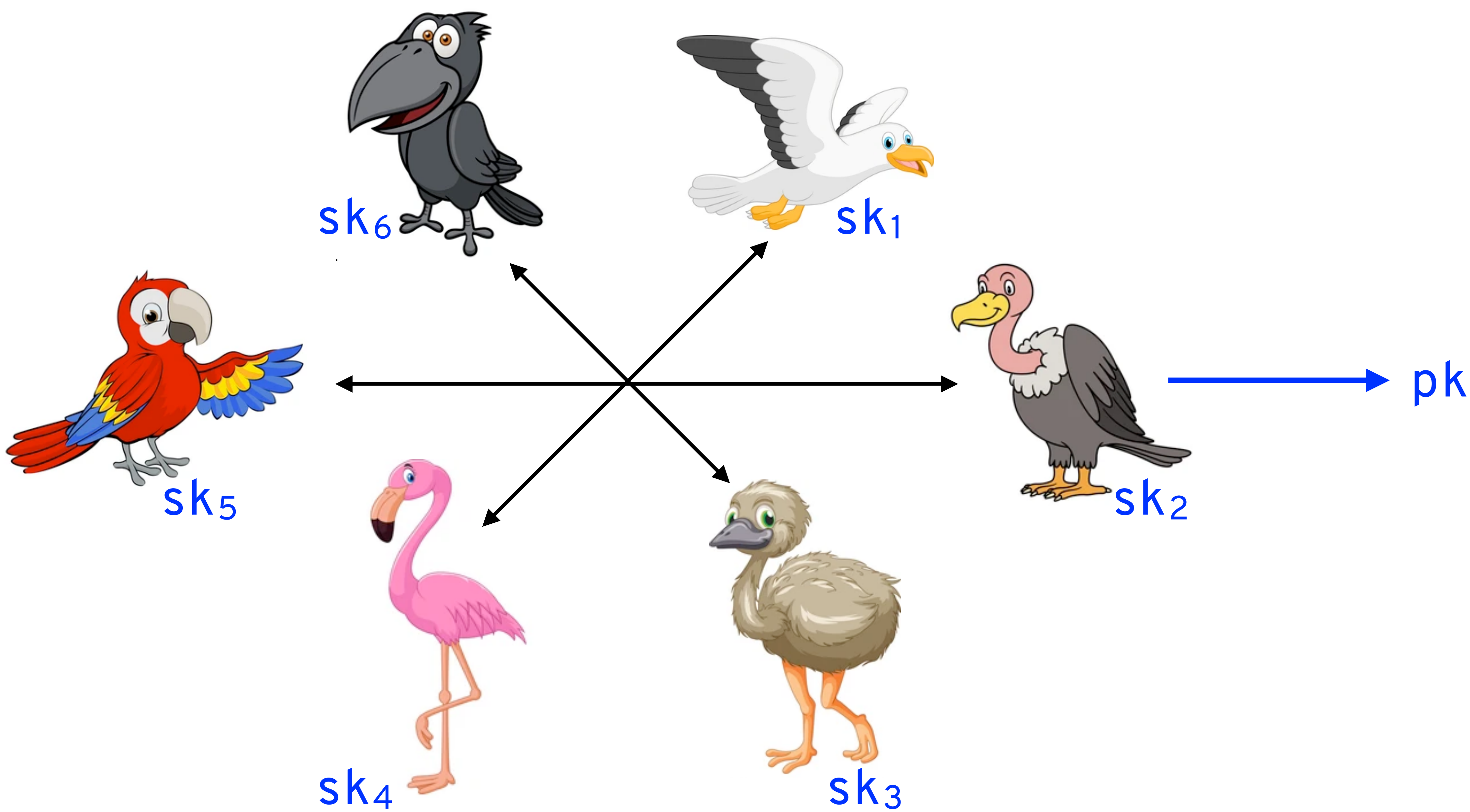
ADAPTIVITY AND ASYNCHRONY IN DISTRIBUTED KEY GENERATION

SARAH MEIKLEJOHN (GOOGLE & UCL)

DISTRIBUTED KEY GENERATION

A distributed key generation (**DKG**) protocol allows a set of participants to generate a threshold public key

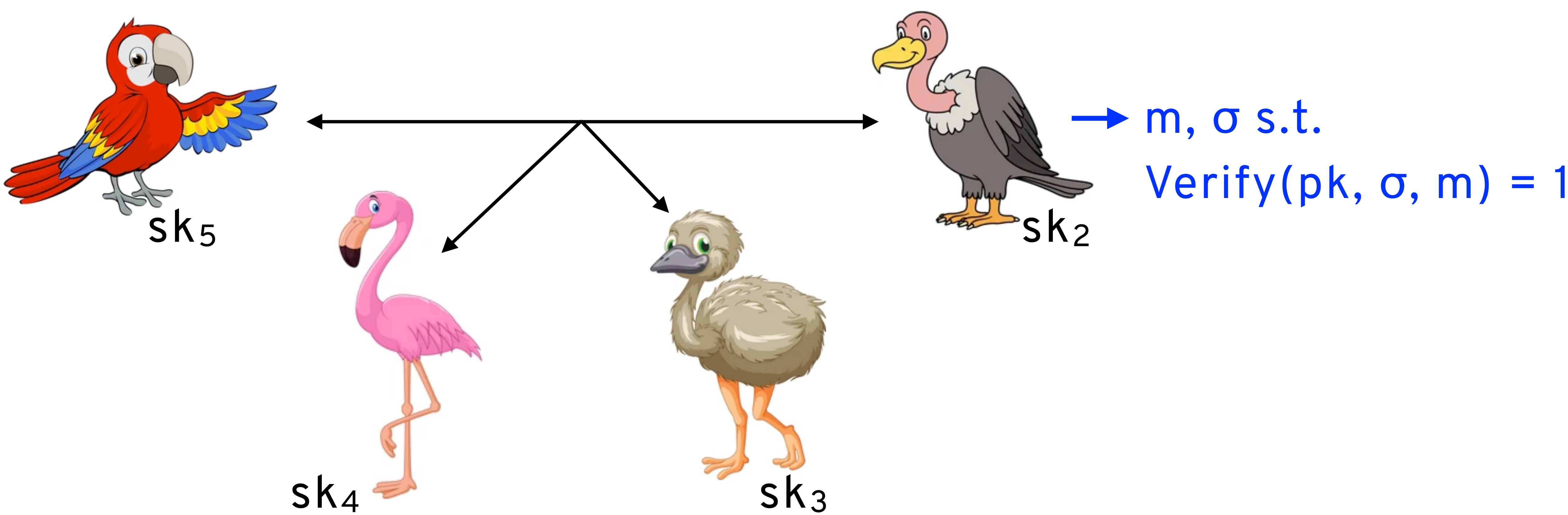


USING A DKG

Classical use cases:

- Want t of n parties to have to collaborate to **decrypt** something
- Want t of n parties to have to collaborate to authorize some action (**sign** something)

For these we expect to run the DKG only once



USING A DKG

Classical use cases:

- Want t of n parties to have to collaborate to **decrypt** something
- Want t of n parties to have to collaborate to authorize some action (**sign** something)

For these we expect to run the DKG **only once**

Can also use DKGs for random beacons

1. Run the DKG to generate a threshold public key
2. Have t parties produce a **unique threshold signature**
3. Hash the unique signature to produce randomness

Here we might run the DKG many times, so there is interest in having efficient DKG protocols that operate in asynchronous environments

ADKG PROTOCOLS

	word complexity	round complexity
[KG10]*	n^4	n
[KMS20]	n^3	n
[APM**M**ST21]	n^3	1
[DYXMKR22]	n^3	$\log(n)$
[GS22]	n^3	1

*assumes partial synchrony

BUILDING A DKG

Most DKGs are based on **secret sharing**

A secret sharing scheme consists of two protocols:

- Deal (Share) allows one party (the **dealer**) to share a secret
- Reconstruct allows t+1 parties to compute the secret

BUILDING A DKG

Most DKGs are based on **secret sharing**

A secret sharing scheme consists of two protocols:

- Deal (Share) allows one party (the **dealer**) to share a secret
- Reconstruct allows t+1 parties to compute the secret

Shamir secret sharing of a secret s:

Deal:

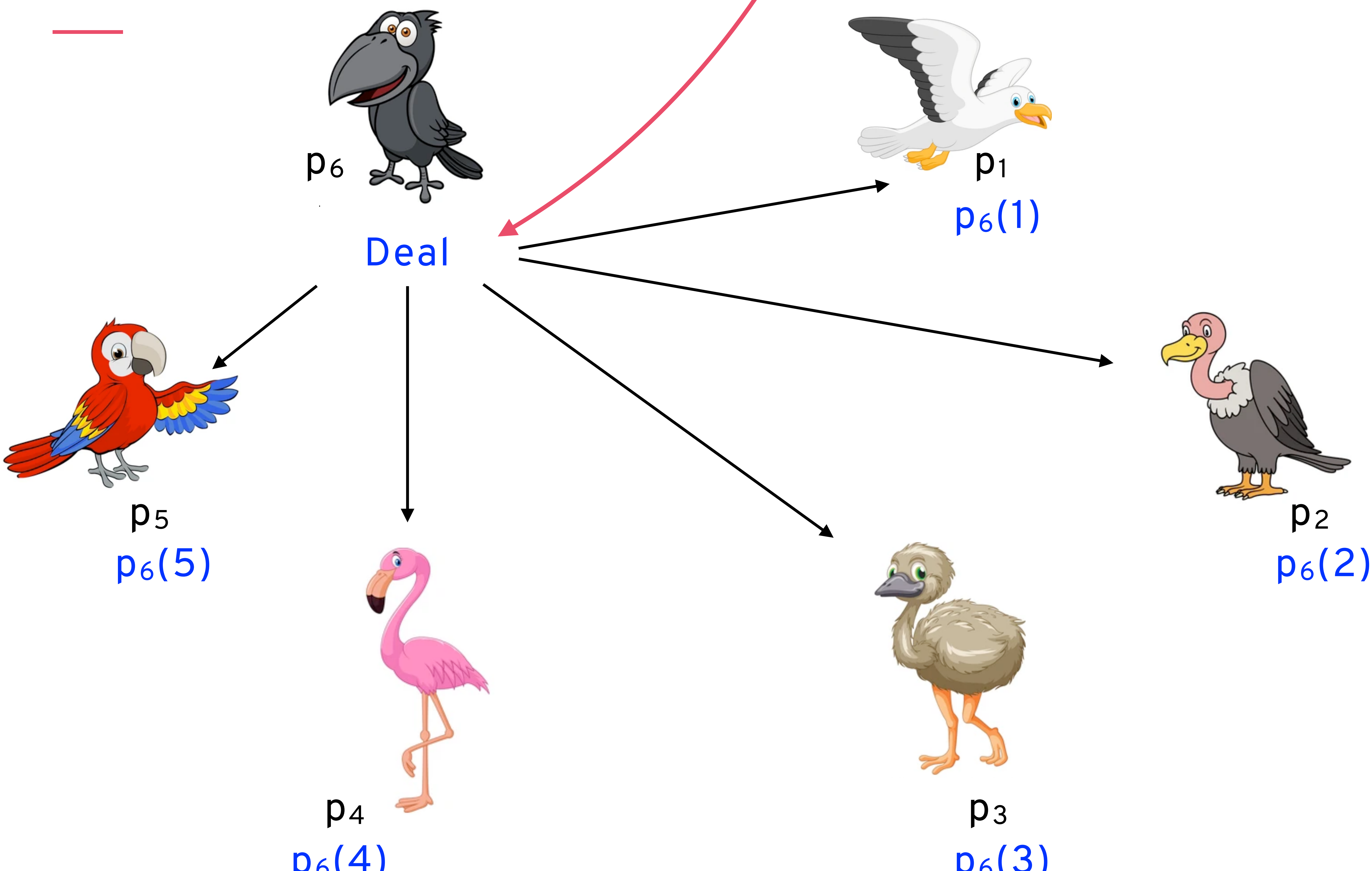
- Form a random degree-t polynomial p(x) such that p(0) = s
- Send p(i) to party i

Reconstruct:

- Party i shares p(i) with other parties
- Once t+1 parties have shared points, can reconstruct p(x) using Lagrange interpolation

A TYPICAL DKG

have everyone act as the dealer in parallel



A TYPICAL DKG

perform reconstruction in the exponent

p_6

$\{p_j(6)\}_j$

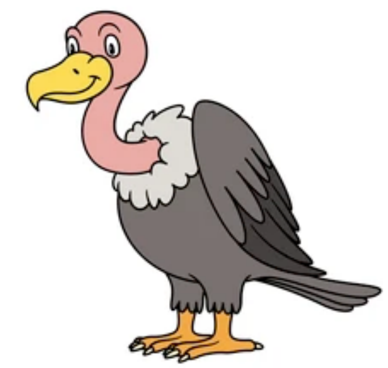
p_1

$\{p_j(1)\}_j$

- party i shares $g^{p_j(i)}$ for all j
- each party interpolates (in the exponent) to get g^{p_j} and computes $\Pi g^{p_j} = g^{\Sigma p_j} = g^p$
- each party evaluates (in the exponent) to compute and output $pk = g^{p(0)}$

p_5

$\{p_j(5)\}_j$



p_2

$\{p_j(2)\}_j$

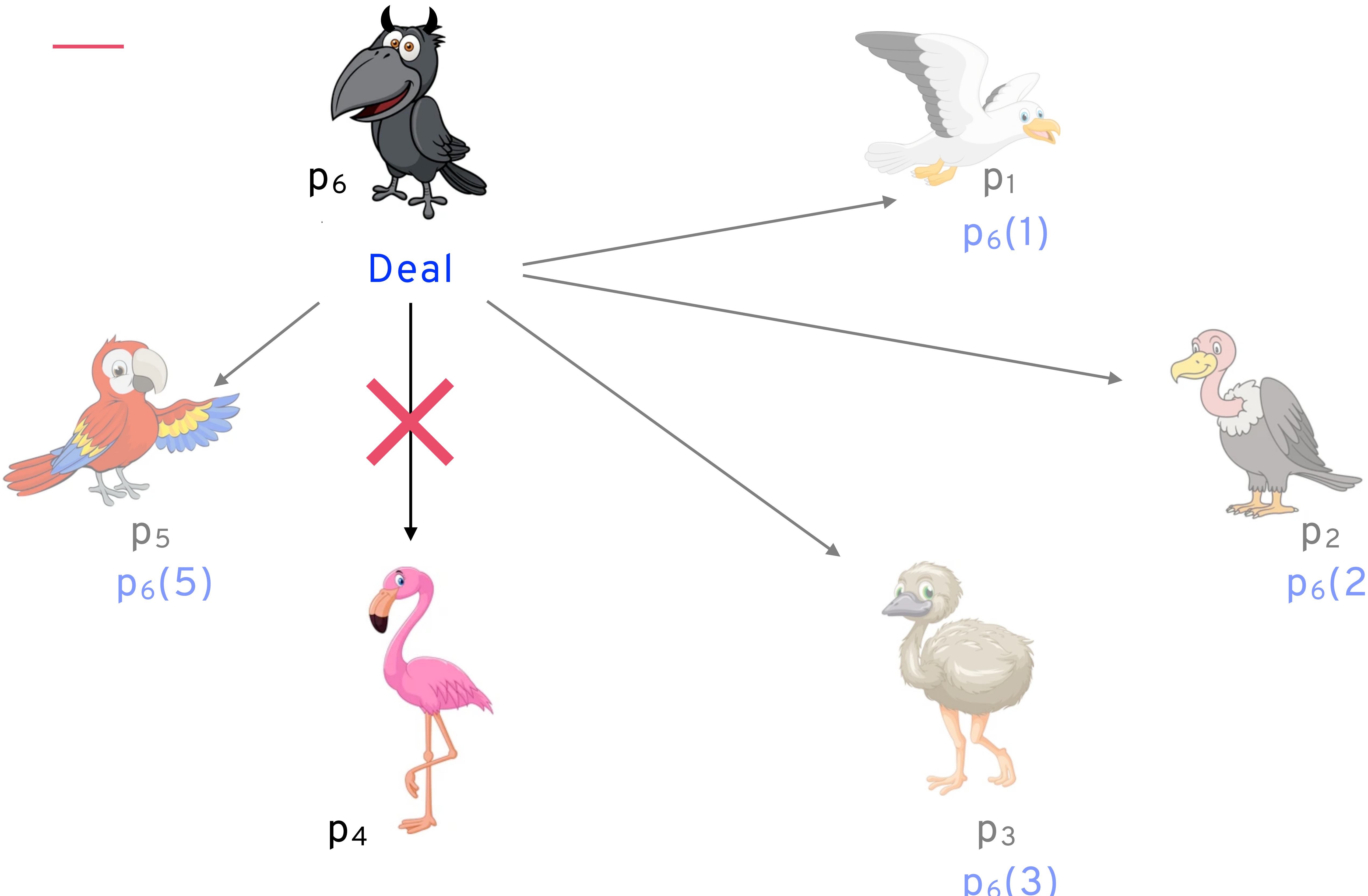
p_4

$\{p_j(4)\}_j$

p_3

$\{p_j(3)\}_j$

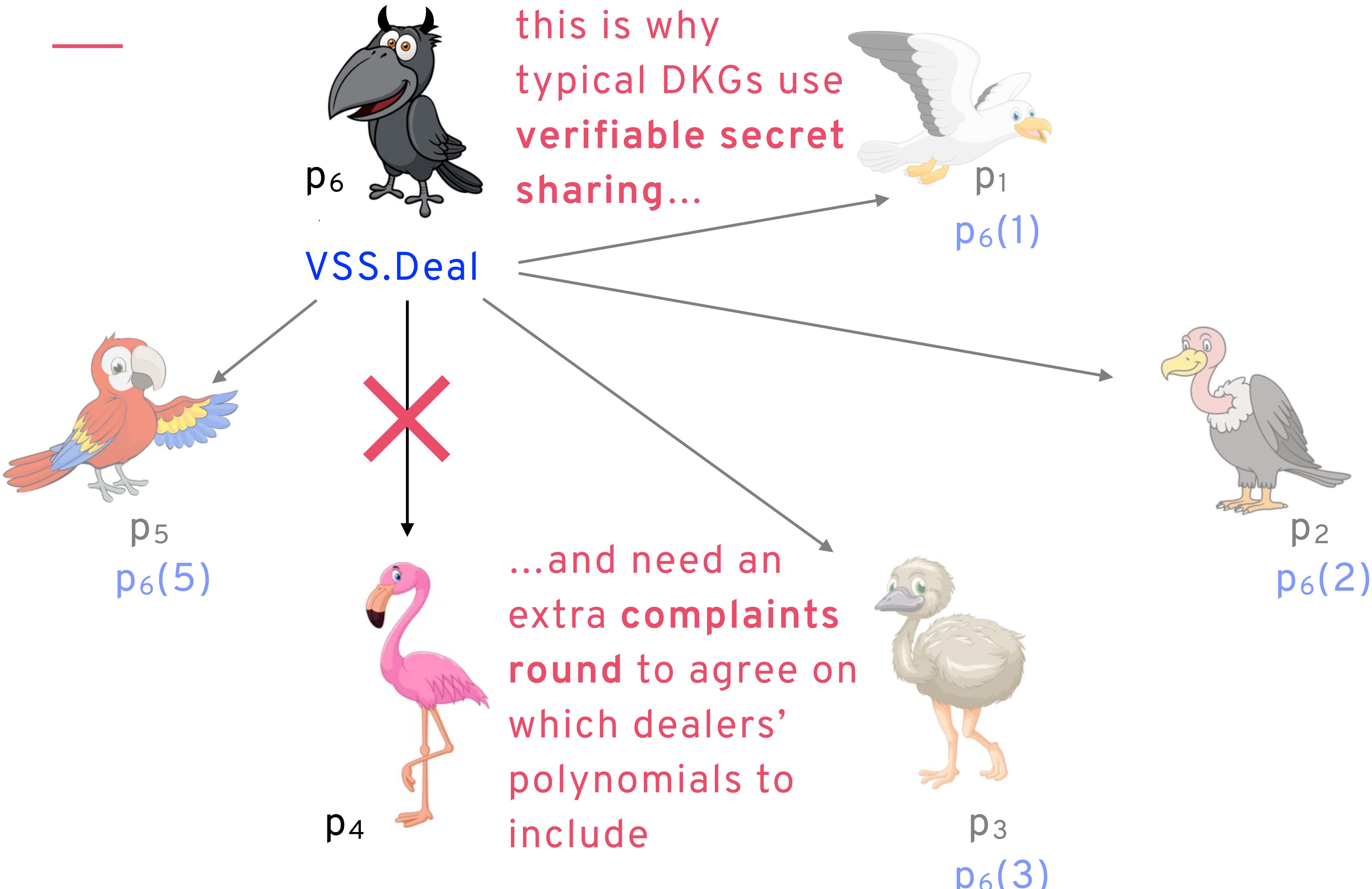
WHEN THINGS GO WRONG



WHEN THINGS GO WRONG

this is why typical DKGs use **verifiable secret sharing...**

...and need an extra **complaints round** to agree on which dealers' polynomials to include



BUILDING A DKG

1. Party i:
   - acts as the VSS dealer
   - participates in VSS sharing for all other parties j
2. All parties agree on a set of dealers D using a complaints round
3. Party i reconstructs, in the exponent, the sum of secrets for dealers in D

so the best way to get a better (A)DKG is to build a better (A)VSS

BINGO [AJM**M**S23]

Bingo is an AVSS that:

- allows secrets to be **packed** (share f+1 secrets with the same complexity as one)
- has **optimal resilience** ($n = 3f + 1$)
- has **$O(n^2)$** word complexity and **$O(1)$** round complexity
- allows for **adaptive corruptions** (new definitions of VSS termination, correctness, and secrecy)

SHARING IN BINGO

- sample $\phi(X, Y)$ s.t. $\phi(-k, 0) = s_k$ for all secrets s_k (**packing**)
- broadcast commitment* to $\phi(X, Y)$
- set $\alpha_i = \phi(X, i)$, $\beta_i = \phi(i, Y)$ (meaning $\boldsymbol{\alpha_i(j) = \beta_j(i)}$)
- send α_i to party i

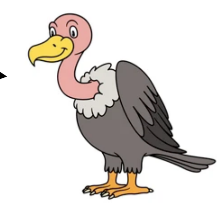
	β_1	β_2	β_3	β_4	β_5
α_1	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$v_{1,4}$	$v_{1,5}$
α_2	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$v_{2,4}$	$v_{2,5}$
α_3	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$	$v_{3,4}$	$v_{3,5}$
α_4	$v_{4,1}$	$v_{4,2}$	$v_{4,3}$	$v_{4,4}$	$v_{4,5}$
α_5	$v_{5,1}$	$v_{5,2}$	$v_{5,3}$	$v_{5,4}$	$v_{5,5}$

the goal in Share is for each party i to learn their α_i polynomial

*using a natural extension of KZG to bivariate polynomials

SHARING IN BINGO

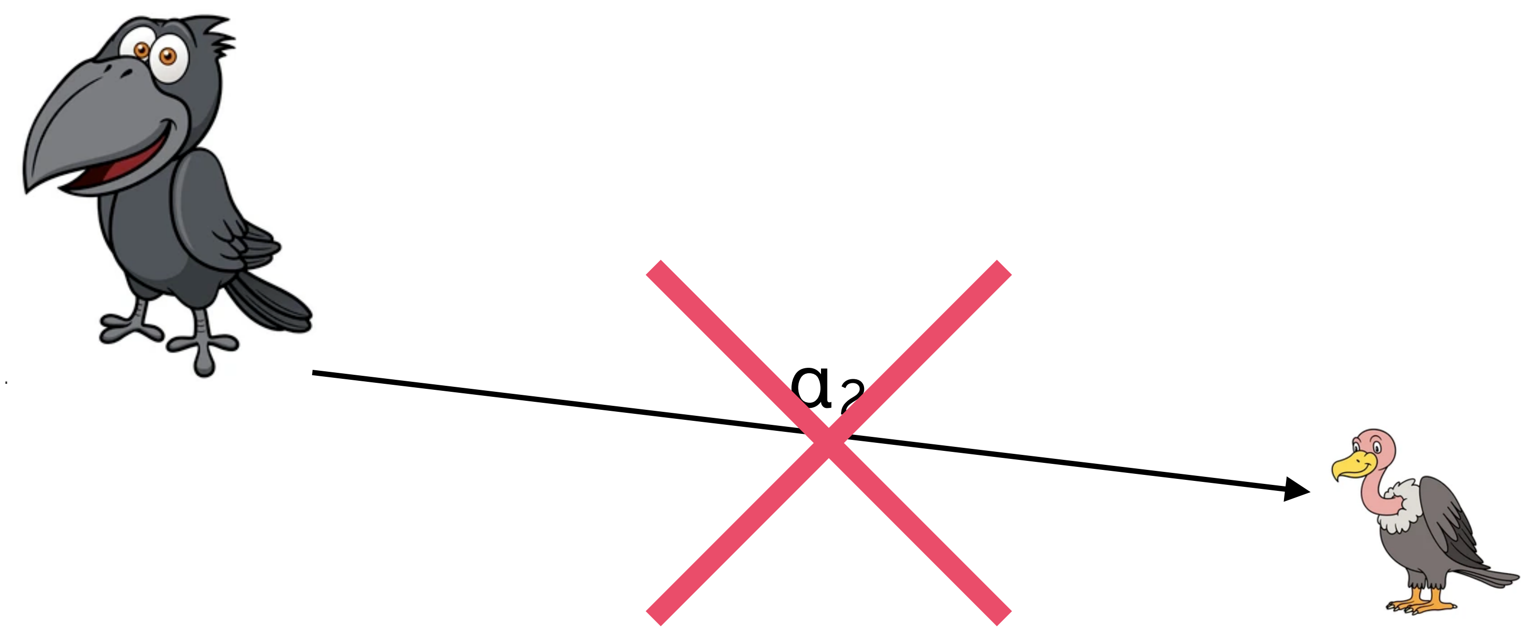
$\xrightarrow{\alpha_2}$



	β_1	β_2	β_3	β_4	β_5
α_1	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$v_{1,4}$	$v_{1,5}$
α_2	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$v_{2,4}$	$v_{2,5}$
α_3	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$	$v_{3,4}$	$v_{3,5}$
α_4	$v_{4,1}$	$v_{4,2}$	$v_{4,3}$	$v_{4,4}$	$v_{4,5}$
α_5	$v_{5,1}$	$v_{5,2}$	$v_{5,3}$	$v_{5,4}$	$v_{5,5}$

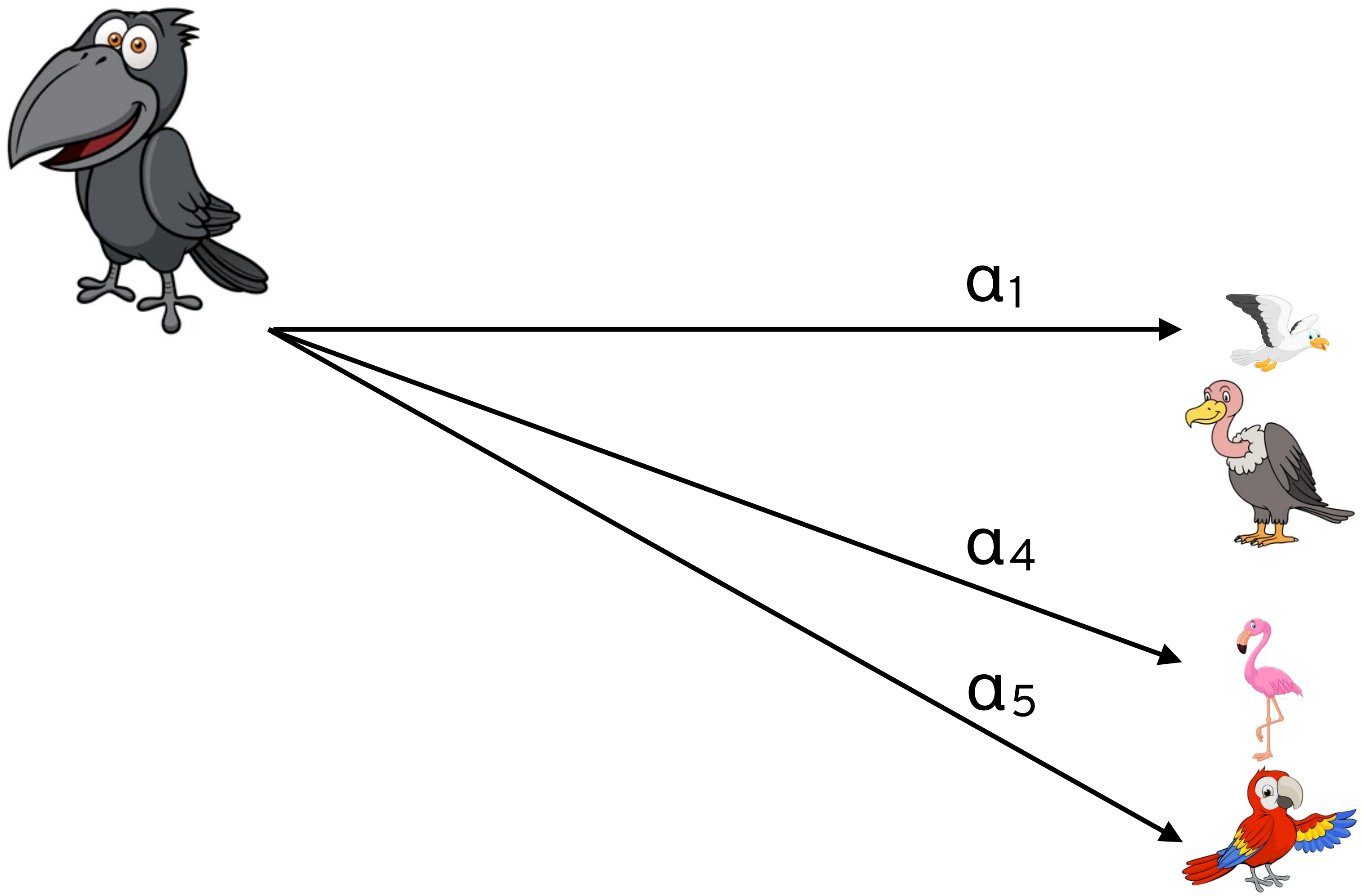
🎉 the goal in Share is for each party i to learn their α_i polynomial

SHARING IN BINGO



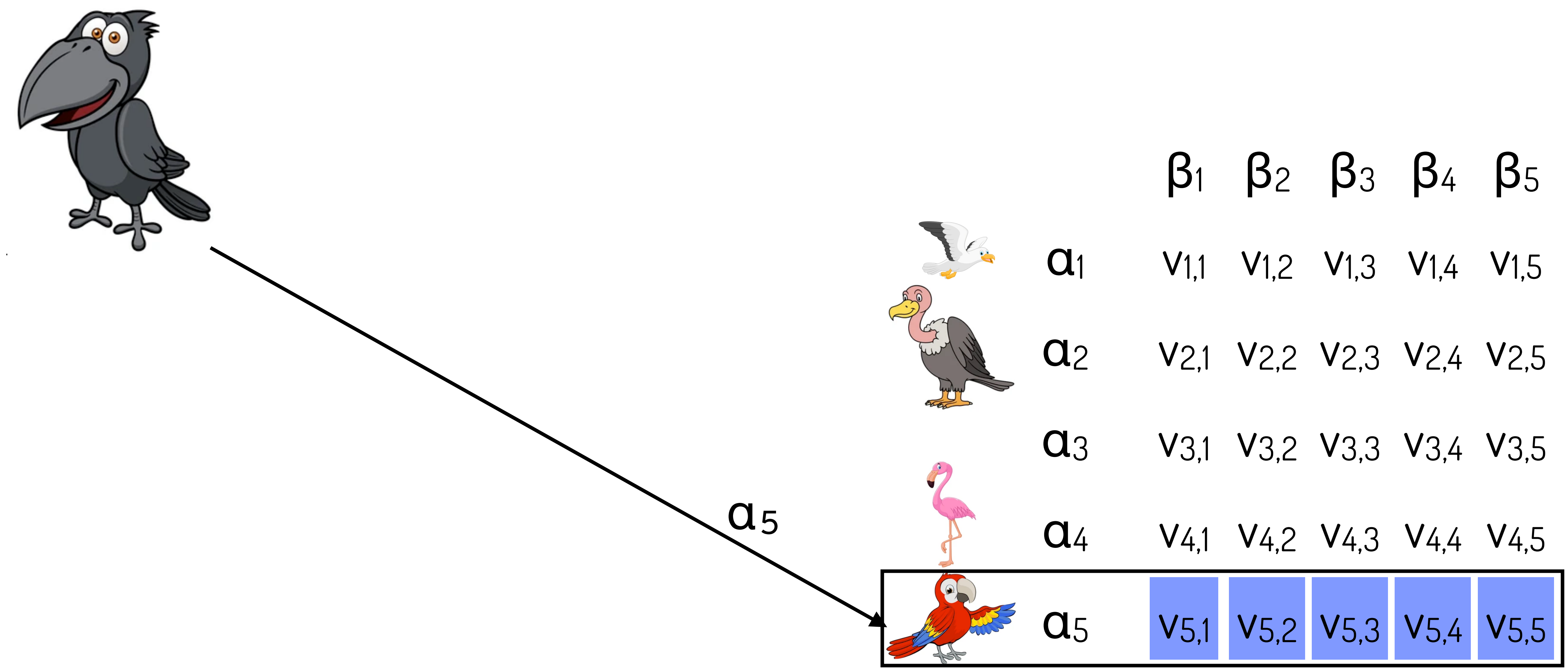
	β_1	β_2	β_3	β_4	β_5
α_1	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$v_{1,4}$	$v_{1,5}$
α_2	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$v_{2,4}$	$v_{2,5}$
α_3	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$	$v_{3,4}$	$v_{3,5}$
α_4	$v_{4,1}$	$v_{4,2}$	$v_{4,3}$	$v_{4,4}$	$v_{4,5}$
α_5	$v_{5,1}$	$v_{5,2}$	$v_{5,3}$	$v_{5,4}$	$v_{5,5}$

SHARING IN BINGO



	β_1	β_2	β_3	β_4	β_5
α_1	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$v_{1,4}$	$v_{1,5}$
α_2	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$v_{2,4}$	$v_{2,5}$
α_3	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$	$v_{3,4}$	$v_{3,5}$
α_4	$v_{4,1}$	$v_{4,2}$	$v_{4,3}$	$v_{4,4}$	$v_{4,5}$
α_5	$v_{5,1}$	$v_{5,2}$	$v_{5,3}$	$v_{5,4}$	$v_{5,5}$

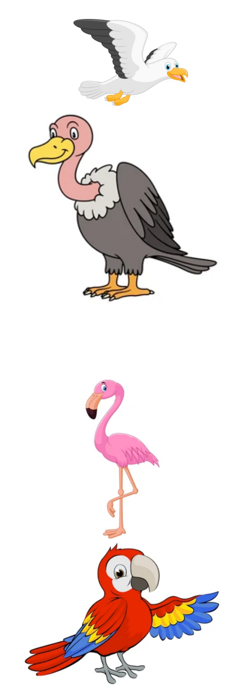
SHARING IN BINGO



send $\alpha_5(j)$ to party j

$\Rightarrow$ party j learns $\beta_j(5)$

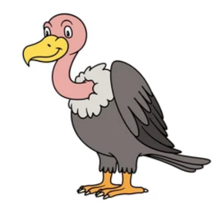
SHARING IN BINGO



$$
\begin{array}{c|ccccc}
 & \beta_1 & \beta_2 & \beta_3 & \beta_4 & \beta_5 \\
\hline
\alpha_1 & \boxed{v_{1,1}} & v_{1,2} & v_{1,3} & v_{1,4} & v_{1,5} \\
\alpha_2 & v_{2,1} & v_{2,2} & v_{2,3} & v_{2,4} & v_{2,5} \\
\alpha_3 & v_{3,1} & v_{3,2} & v_{3,3} & v_{3,4} & v_{3,5} \\
\alpha_4 & \boxed{v_{4,1}} & v_{4,2} & v_{4,3} & v_{4,4} & v_{4,5} \\
\alpha_5 & \boxed{v_{5,1}} & v_{5,2} & v_{5,3} & v_{5,4} & v_{5,5}
\end{array}
$$

given enough points, party 1 can interpolate to learn β_1

SHARING IN BINGO



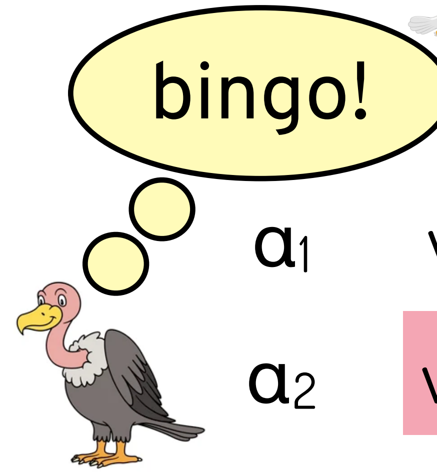
	β_1	β_2	β_3	β_4	β_5
α_1	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$v_{1,4}$	$v_{1,5}$
α_2	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$v_{2,4}$	$v_{2,5}$
α_3	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$	$v_{3,4}$	$v_{3,5}$
α_4	$v_{4,1}$	$v_{4,2}$	$v_{4,3}$	$v_{4,4}$	$v_{4,5}$
α_5	$v_{5,1}$	$v_{5,2}$	$v_{5,3}$	$v_{5,4}$	$v_{5,5}$

send $\beta_1(j)$ to party j

$\Rightarrow$ party j learns $\alpha_j(1)$

SHARING IN BINGO

🎉 given enough points, party 2 can interpolate to learn α_2!

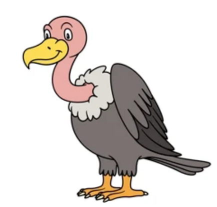


	β_1	β_2	β_3	β_4	β_5
α_1	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$v_{1,4}$	$v_{1,5}$
α_2	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$v_{2,4}$	$v_{2,5}$
α_3	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$	$v_{3,4}$	$v_{3,5}$
α_4	$v_{4,1}$	$v_{4,2}$	$v_{4,3}$	$v_{4,4}$	$v_{4,5}$
α_5	$v_{5,1}$	$v_{5,2}$	$v_{5,3}$	$v_{5,4}$	$v_{5,5}$

SHARING IN BINGO

Some hidden complexities:

- Parties have to prove correctness of their evaluations (more work for the commitment)
- All parties need to have the same commitment (use **reliable broadcast** [DXR21])
- Adaptivity!



	β_1	β_2	β_3	β_4	β_5
α_1	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$v_{1,4}$	$v_{1,5}$
α_2	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$v_{2,4}$	$v_{2,5}$
α_3	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$	$v_{3,4}$	$v_{3,5}$
α_4	$v_{4,1}$	$v_{4,2}$	$v_{4,3}$	$v_{4,4}$	$v_{4,5}$
α_5	$v_{5,1}$	$v_{5,2}$	$v_{5,3}$	$v_{5,4}$	$v_{5,5}$

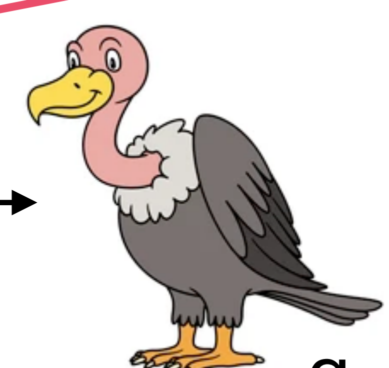
RECONSTRUCTION IN BINGO

- sample $\phi(X, Y)$ s.t. $\phi(-k, 0) = s_k$ for all secrets s_k (**packing**)
    - party j shares $\alpha_j(-k)$
    - interpolates β_{-k}
    - evaluates $s_k = \beta_{-k}(0)$

can reconstruct one secret at a time

same old trick ($\alpha_j(-k) = \beta_{-k}(j)$)

α_5



α_2

α_4

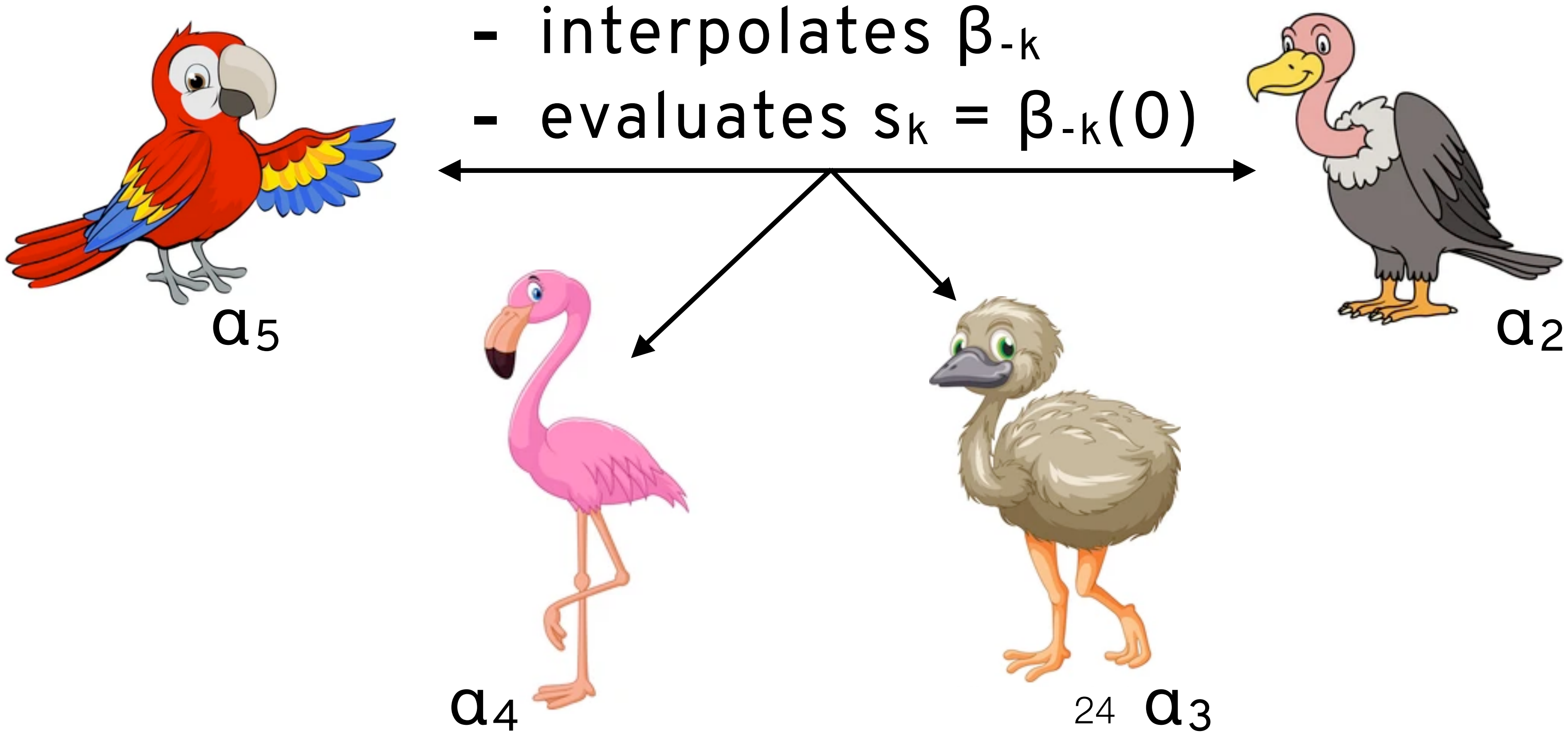
α_3

RECONSTRUCTION IN BINGO

can also reconstruct sums of secrets with the same complexity!

- party j gets $\alpha_{j,i}$ when i is dealing
- computes $\alpha_j = \Sigma\alpha_{j,i}$
- shares $\alpha_j(-k)$
- interpolates β_{-k}
- evaluates $s_k = \beta_{-k}(0)$

(same for the batch reconstruction of multiple secrets)



BUILDING A DKG

1. Party i:
   - acts as the **VSS dealer**
   - participates in **VSS sharing** for all other parties j
2. All parties agree on a set of dealers D using a complaints round
3. Party i **reconstructs, in the exponent**, the sum of secrets for dealers in D

BUILDING A DKG

1. Party i:
   - acts as the Bingo dealer
   - participates in Bingo sharing for all other parties j
2. All parties agree on a set of dealers D using a complaints round
3. Party i reconstructs, in the exponent, the sum of secrets for dealers in D
   - computes $\alpha_j = \Sigma\alpha_{j,i}$ for j in D
   - shares $g^{\alpha j(-k)}$
   - interpolates and evaluates to output pk = $g^{\beta\text{-}k(0)}$

can do this by sending one point rather than O(n)

BUILDING A DKG

1. Party i:
   - acts as the Bingo dealer
   - participates in Bingo sharing for all other parties j
2. All parties agree on a set of dealers D using a ~~complaints round~~ **VABA**
3. Party i reconstructs, in the exponent, the sum of secrets for dealers in D
   - computes $\alpha_j = \Sigma\alpha_{j,i}$ for j in D
   - shares $g^{\alpha_j(-k)}$
   - interpolates and evaluates to output $pk = g^{\beta_{-k}(0)}$

VABA

Validated asynchronous Byzantine agreement (VABA) allows parties to agree on a valid value

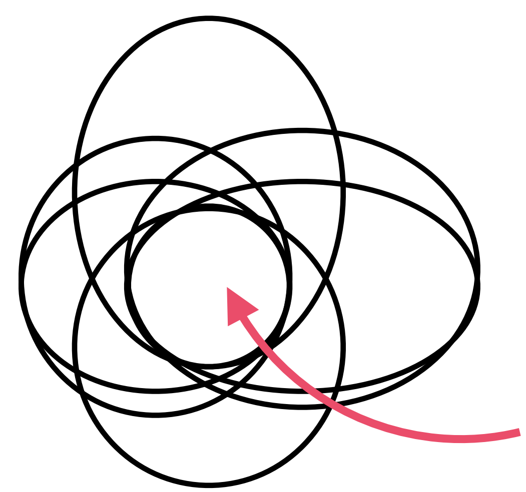
- all non-faulty parties complete the protocol and output the same value
- this value is valid according to some external validity function checkValidity

For us, checkValidity(dealers, sigs) outputs 1 iff:

- $|\text{dealers}| \geq f+1$
- $|\text{sigs}| \geq f+1$
- Verify(pk_j, σ_j, dealers) for all (j, σ_j) in sigs

VABA [AJMMST21]

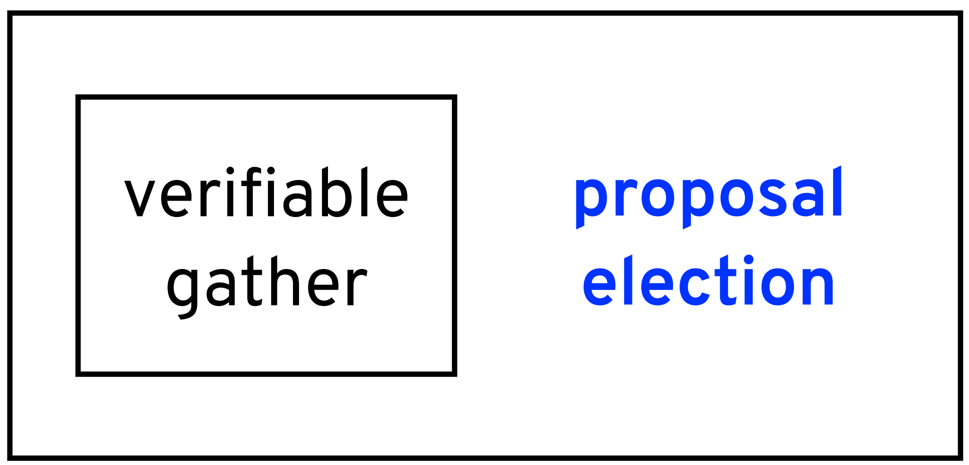
verifiable gather

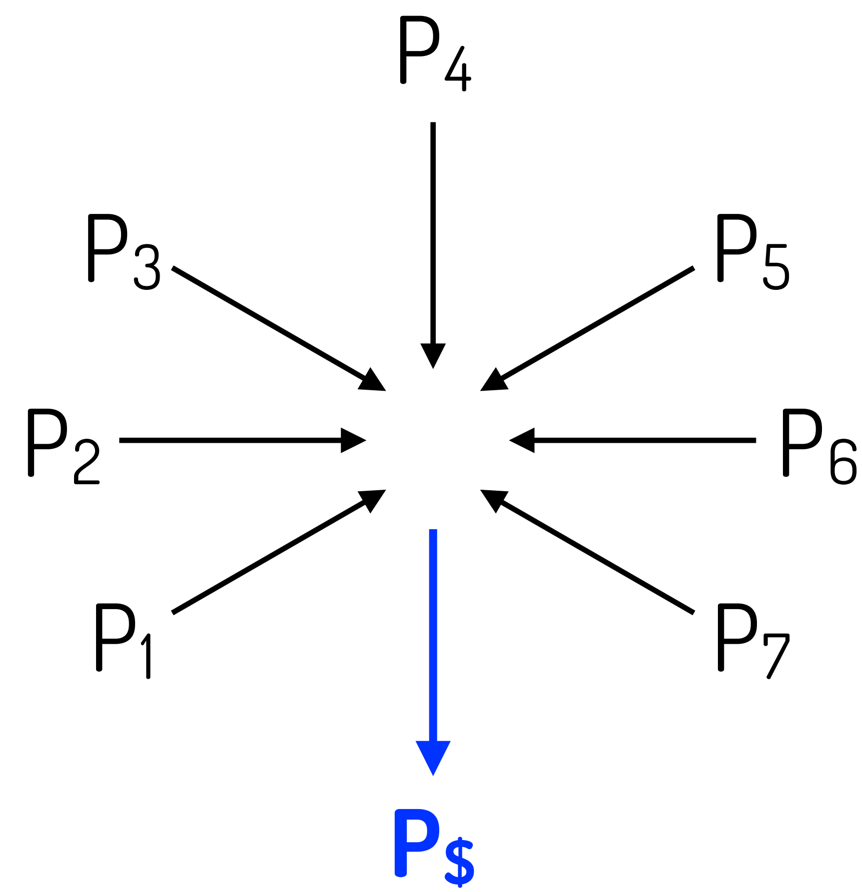


any set that passes verification must be a super-set of this common **core**

built this based on reliable broadcast

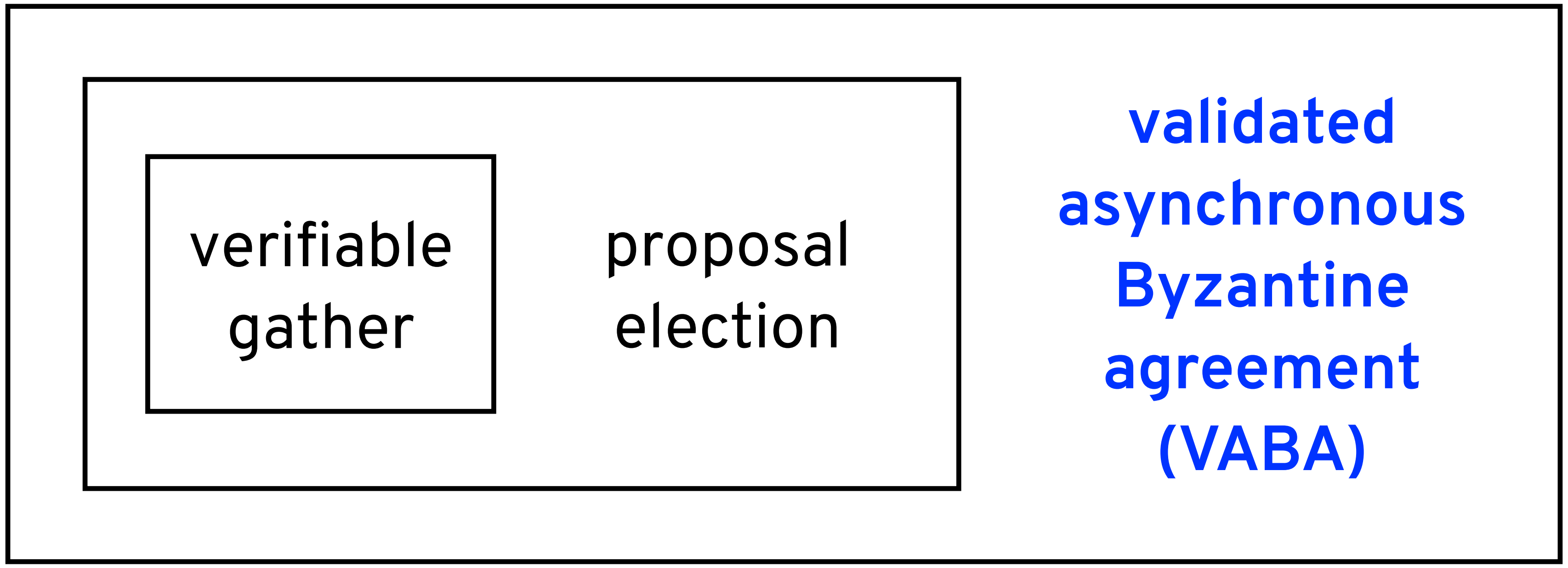
VABA [AJM**M**ST21]





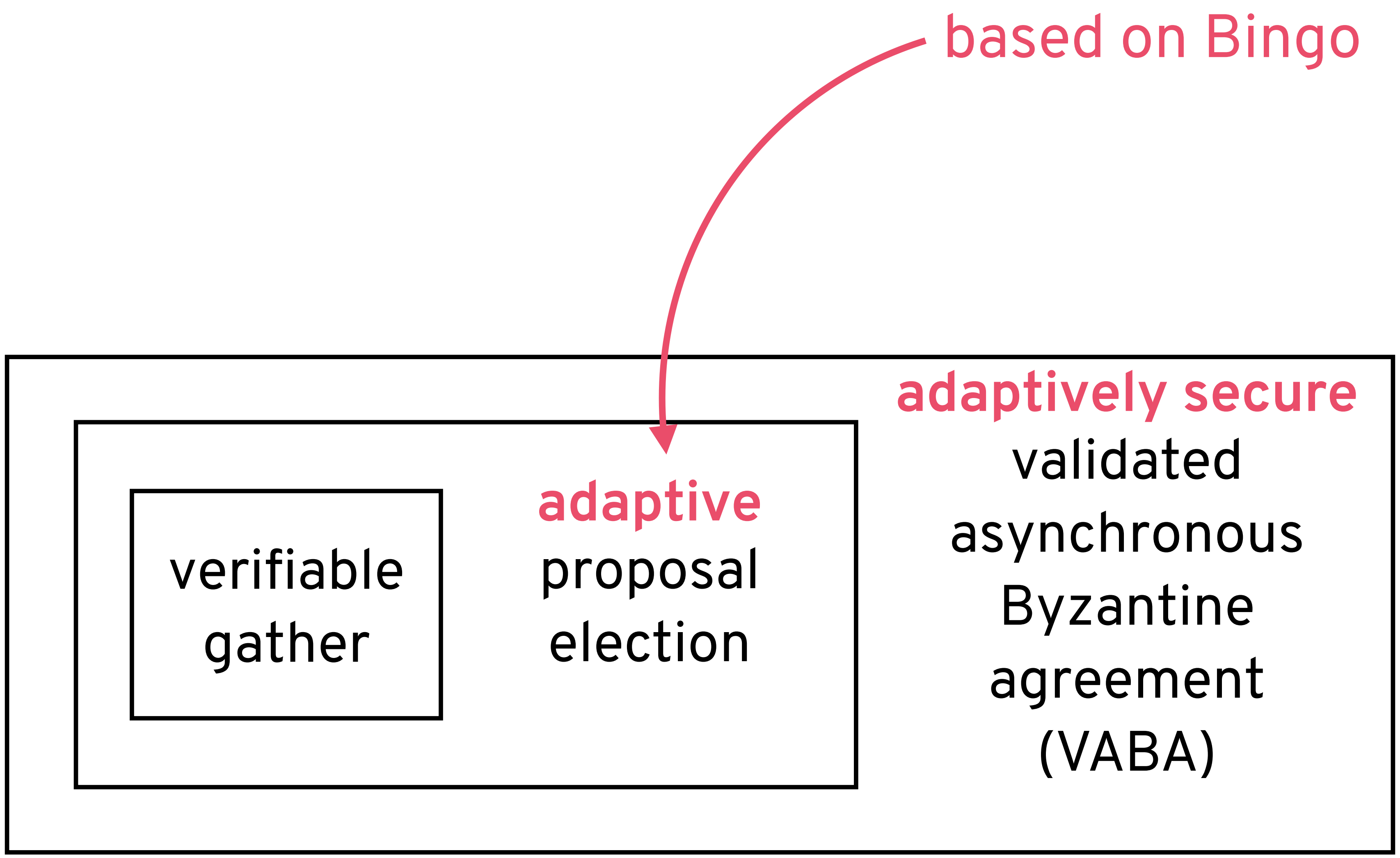
built this based on threshold VRFs and verifiable gather

VABA [AJM**M**ST21]



built this ("No Waitin' Hotstuff") based on proposal election

ADAPTIVELY SECURE VABA



ADKG PROTOCOLS

	word complexity	round complexity	trusted setup?	high threshold?	adaptive
[KG10]*	n^4	n	no	no	no
[KMS20]	n^3	n	no	yes	no
[APM**M**ST21]	n^3	1	no	no	no
[DYXMKR22]	n^3	log(n)	no	yes	no
[GS22]	n^3	1	no	no	no
Our work	n^3	1	yes	yes	yes

*assumes partial synchrony

THANKS!

ANY QUESTIONS?