# Knowledge and Common Knowledge in a Distributed Environment

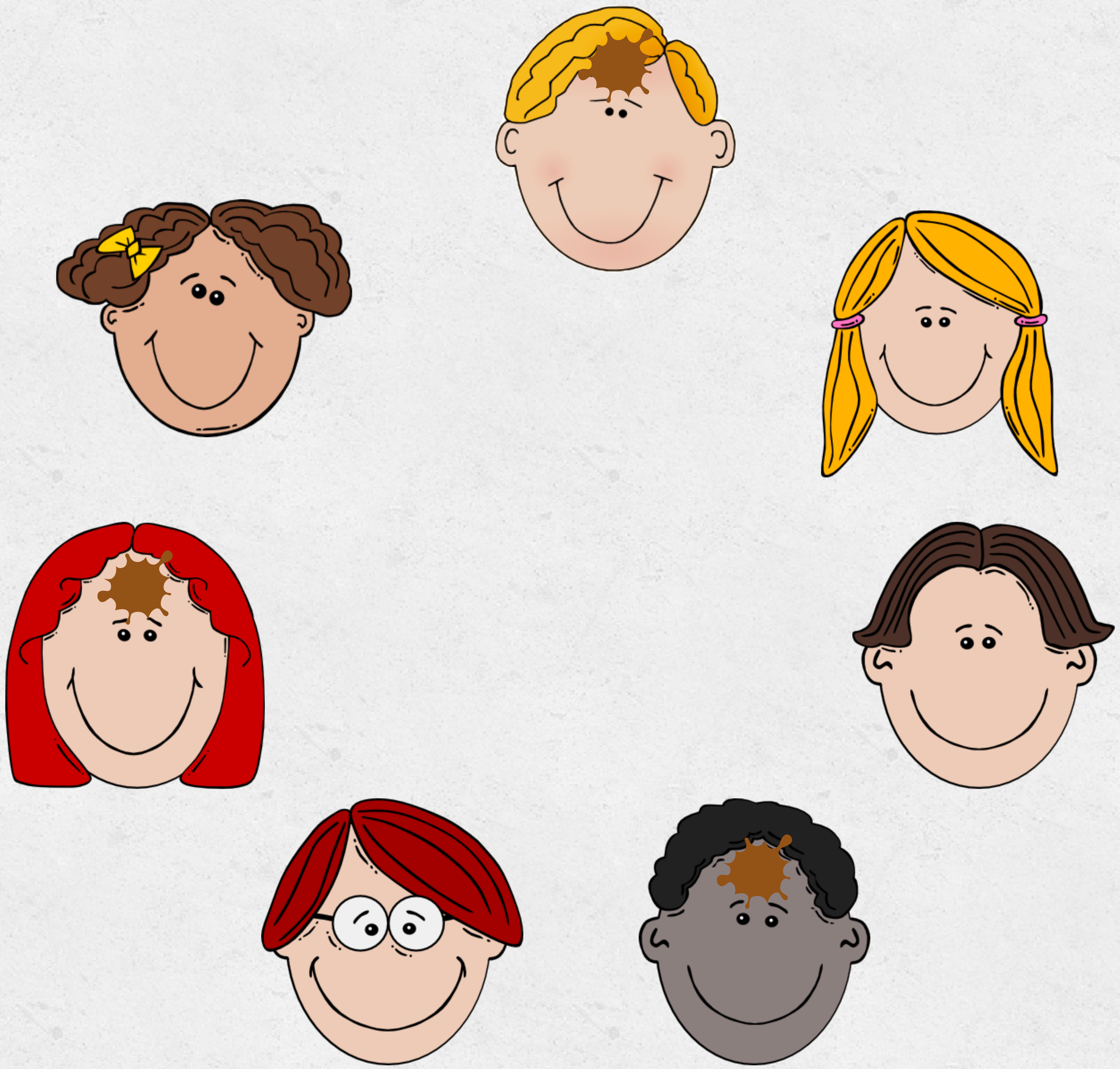
Ellis Michael

# Administrivia

- Everyone should have a GitLab repo. If not, email me.
- Everyone should have signed up for Piazza.
- Lab 1 due in 1 week.

# Muddy Foreheads

- $n$ children, $k$ get mud on their foreheads
- Children sit in circle.
- Teacher announces, "Someone has mud on their forehead."
- Teacher repeatedly asks, "Raise your hand if  you know you have mud on your forehead."
- What happens?



# Muddy Foreheads

- $n$ children, $k$ get mud on their foreheads
- Children sit in circle.
- Teacher announces, "Someone has mud on their forehead."
- Teacher repeatedly asks, "Raise your hand if you know you have mud on your forehead."
- What happens?

# Muddy Foreheads

**Answer:** On the $k$th round, all of the muddy children know they have mud on their forehead, raise their hands.

**"Proof" by induction on $k$.** When $k$=1, the muddy child knows no other child is muddy, must be muddy themself.
When k=2, on the first round, both muddy children see each other, cannot conclude they themselves are muddy. But after neither raises their hand, they realize there must be two muddy children, raise their hand.

In general, when $k$>1, after round $k$-1, if there were $k$-1 muddy foreheads, all of those children would have raised their hands (by induction). Therefore, each muddy child knows they're muddy and raises their hand on the $k$th round.

# The Muddy Forehead "Paradox"

*If $k>1$, the teacher didn't say anything anyone didn't already know!*

# Knowledge and Common Knowledge

- $\varphi$ — a fact (e.g. "someone has mud on their forehead")
- $K_i\varphi$ — agent $i$ knows $\varphi$

  $K_i\varphi \supset \varphi$ (knowledge of $\varphi$ implies $\varphi$)
- $D_G\varphi$ — group $G$ has distributed knowledge of $\varphi$
- $S_G\varphi$ — someone in group $G$ knows $\varphi$

- $E_G\varphi = E_G{}^1\varphi$ — everyone knows $\varphi$
- $E_G{}^{k+1}\varphi$ — everyone knows $E_G{}^k\varphi$
- $C_G\varphi$ — $\varphi$ is common knowledge in $G$ (everyone knows everyone knows everyone knows everyone knows... $\varphi$)

$C_G\varphi \supset \ldots \supset E_G{}^{k+1}\varphi \supset \ldots \supset E_G\varphi \supset S_G\varphi \supset D_G\varphi \supset \varphi$

# An Example

You and your friends want to decide where to eat lunch.

- $\varphi =$ "we will go to Chipotle"
- $\forall p \in G : p$ wants to go to Chipotle $\supset D_G\varphi$
- You all tell your preferences to Alice privately. Then $K_{\text{Alice}}\varphi$ holds (implies $S_G\varphi$)
- Alice then tells everyone the result privately. $E_G\varphi$ holds.
- Alice then tells everyone that she told everyone, $E_G{}^2\varphi$. etc.
- Alice announces the result publicly in front of the group, $C_G\varphi$.

# Muddy Foreheads and Common Knowledge

The muddy foreheads argument implicitly requires $E^{k}\varphi$, where $\varphi$ is "someone has mud on their forehead" and $k$ is the number of muddy foreheads.

At the beginning, children only have $E^{k-1}\varphi$.

When the teacher spoke in plain sight, imparted *common knowledge* (i.e., $C\varphi \supset E^{k}\varphi$).

# A Twist

What if, instead of announcing "someone has mud on their forehead," the teacher pulled each student aside and told them privately?

# Twist 2: Electric Boogaloo

Teacher pulls each student aside.

Each student, unbeknownst to the others, planted listening devices on all other students, heard the teacher tell the other students "someone has mud on their forehead"?

# What Do They Really Need to Know?

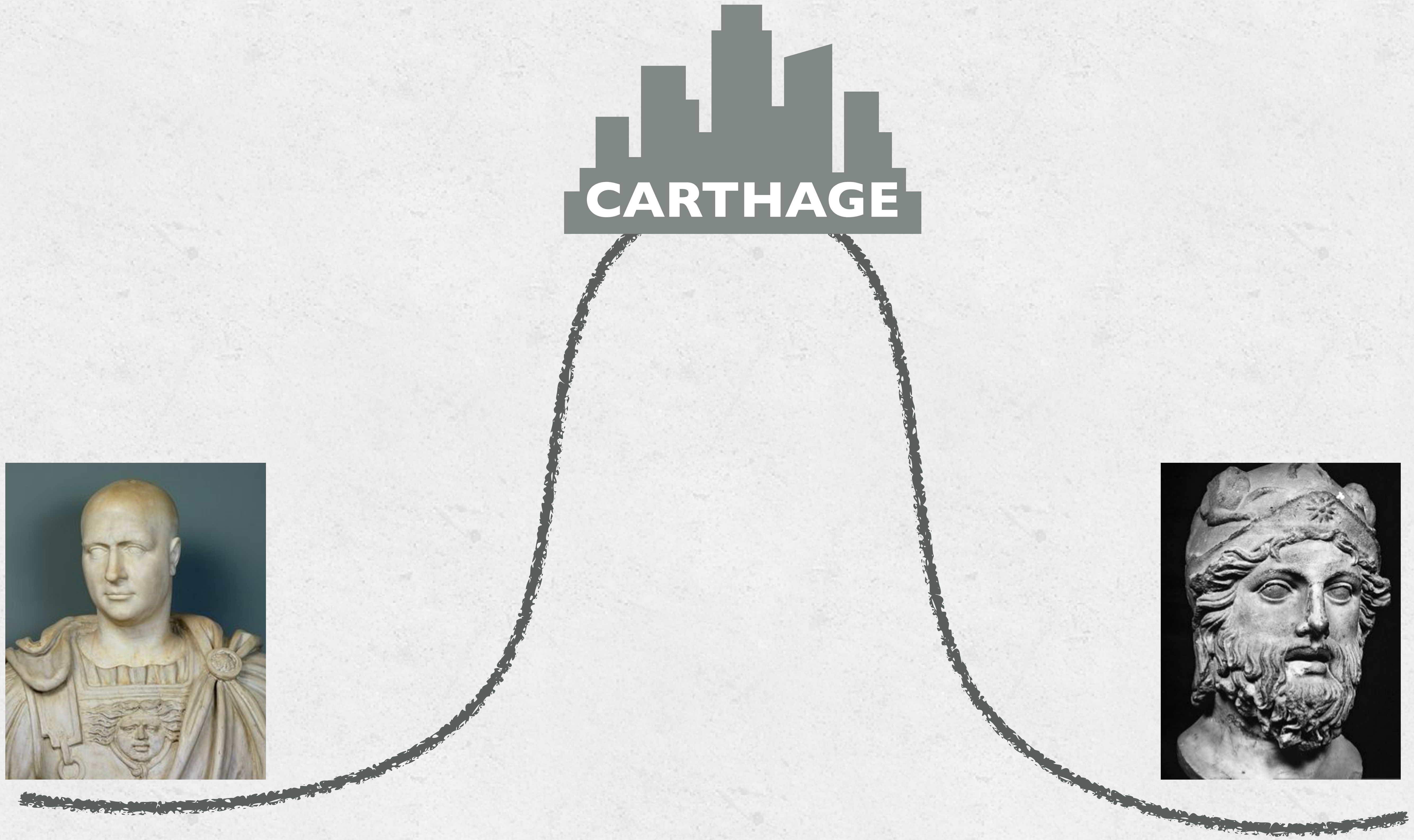
Fun problems:

- What is the weakest level of knowledge needed by the children such that one or all of the muddy ones will eventually raise their hand?
- Is it possible for one muddy child to raise their hand and another to never realize their forehead is muddy?

Come to office hours!

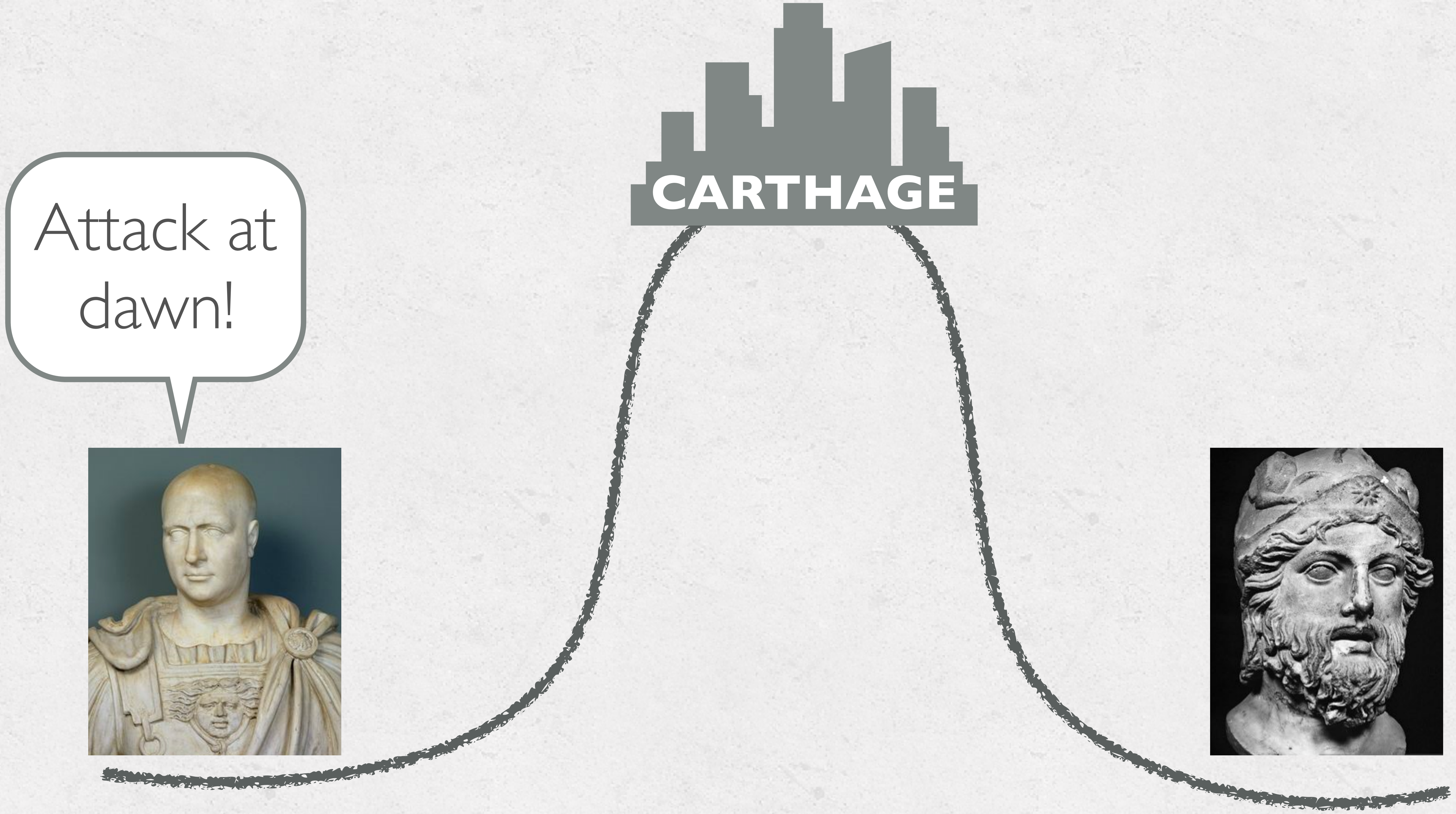
# Coordinated Attack

- Two generals, on opposite sides of a city on a hill.
- If they attack simultaneously, they will be victorious. If one attacks without the other, they will both be defeated.
- Can communicate by messenger. Messengers can get lost or be captured.
- How do they ensure they can take the city?



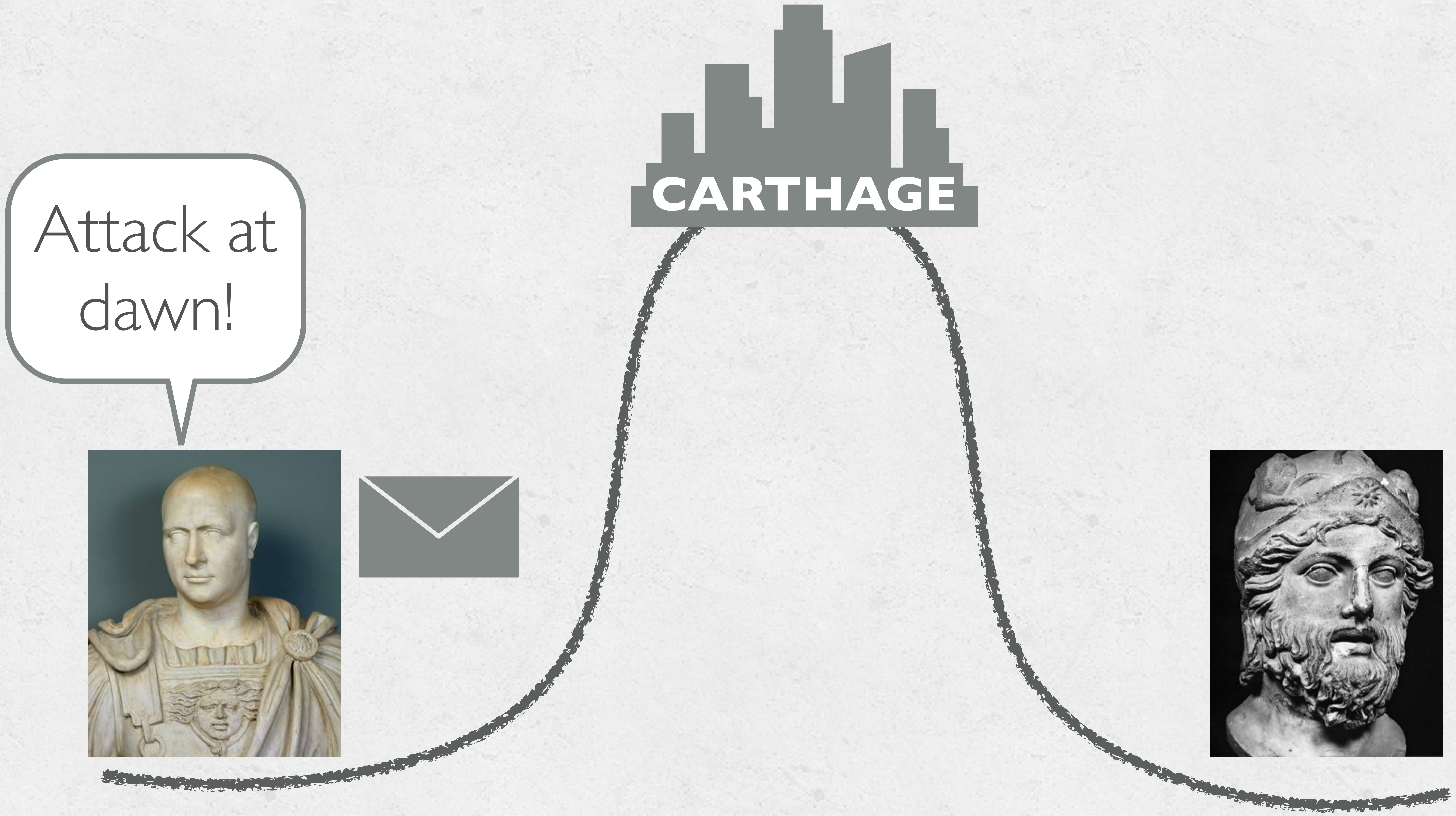
# Coordinated Attack

- Two generals, on opposite sides of a city on a hill.
- If they attack simultaneously, they will be victorious. If one attacks without the other, they will both be defeated.
- Can communicate by messenger. Messengers can get lost or be captured.
- How do they ensure they can take the city?



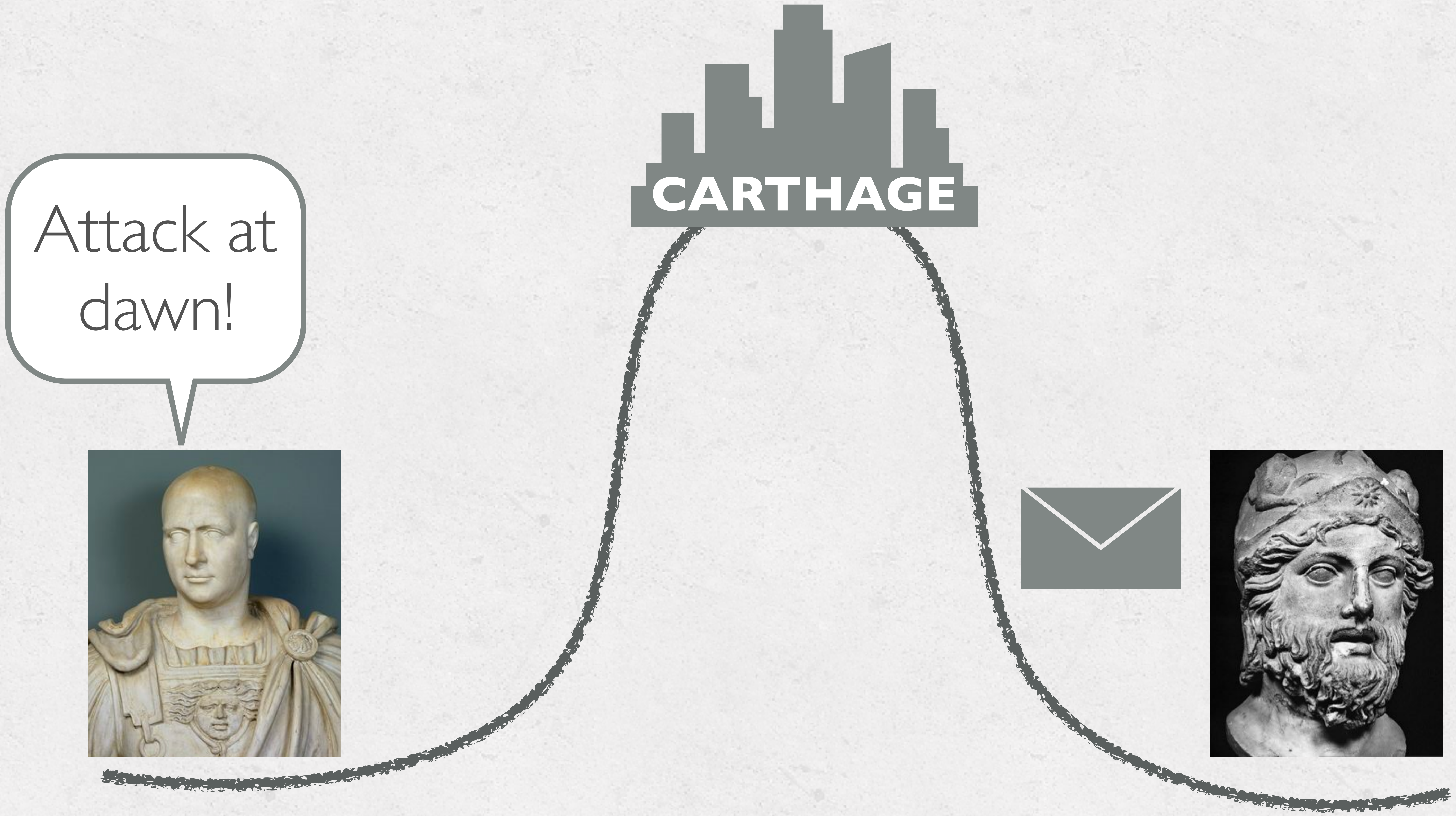
# COORDINATED ATTACK

- Two generals, on opposite sides of a city on a hill.
- If they attack simultaneously, they will be victorious. If one attacks without the other, they will both be defeated.
- Can communicate by messenger. Messengers can get lost or be captured.
- How do they ensure they can take the city?



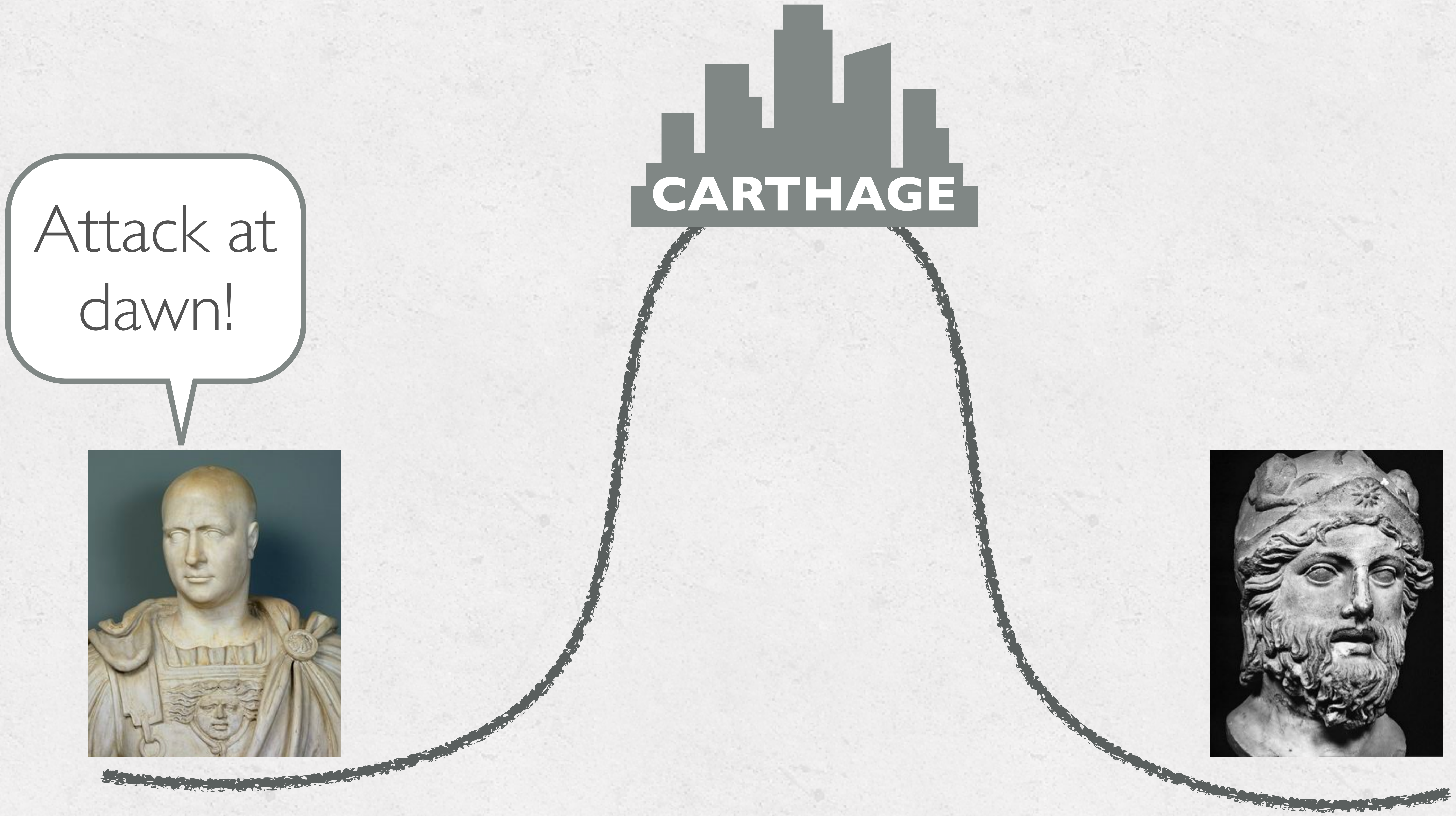
# COORDINATED ATTACK

- Two generals, on opposite sides of a city on a hill.
- If they attack simultaneously, they will be victorious. If one attacks without the other, they will both be defeated.
- Can communicate by messenger. Messengers can get lost or be captured.
- How do they ensure they can take the city?



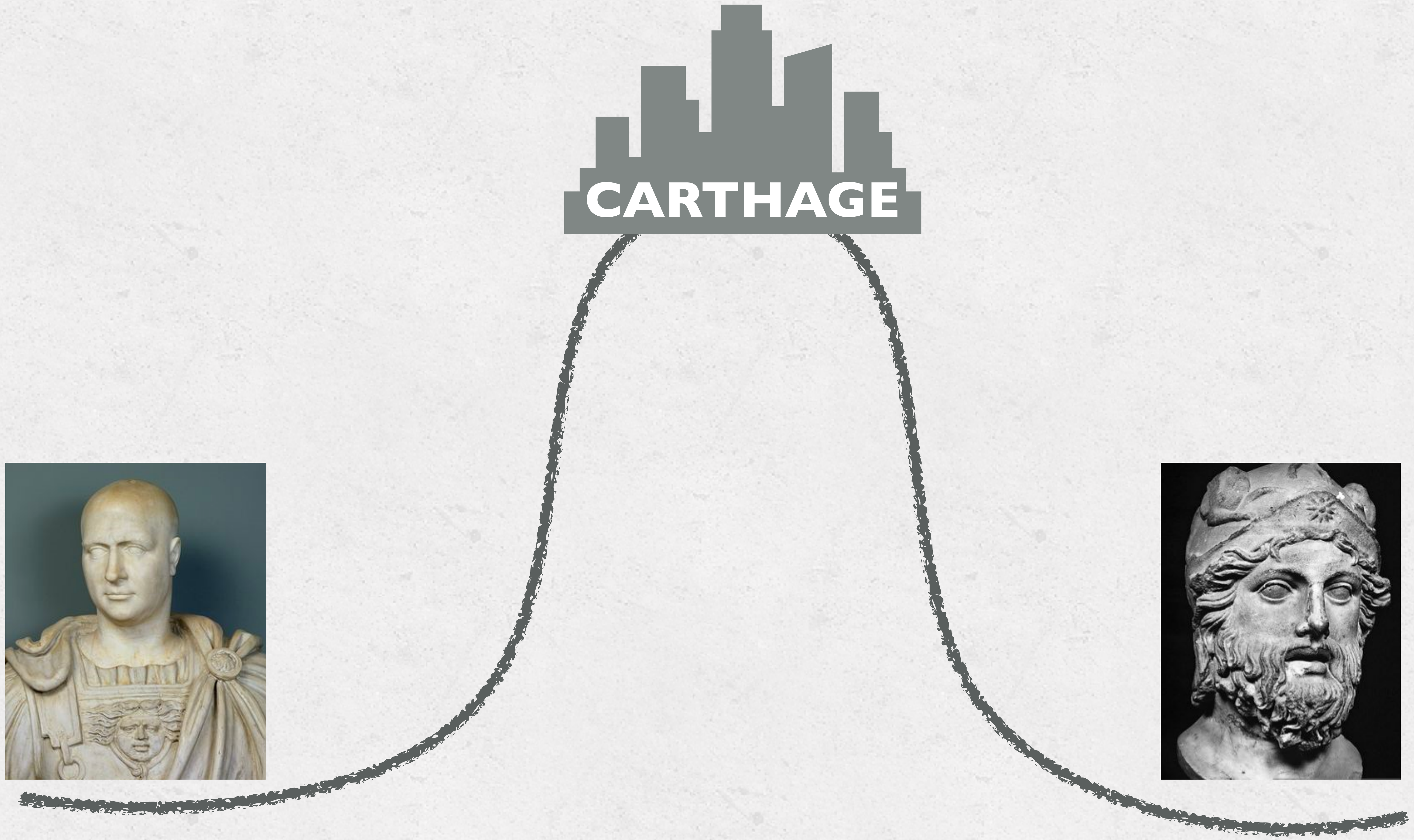
# COORDINATED ATTACK

- Two generals, on opposite sides of a city on a hill.
- If they attack simultaneously, they will be victorious. If one attacks without the other, they will both be defeated.
- Can communicate by messenger. Messengers can get lost or be captured.
- How do they ensure they can take the city?



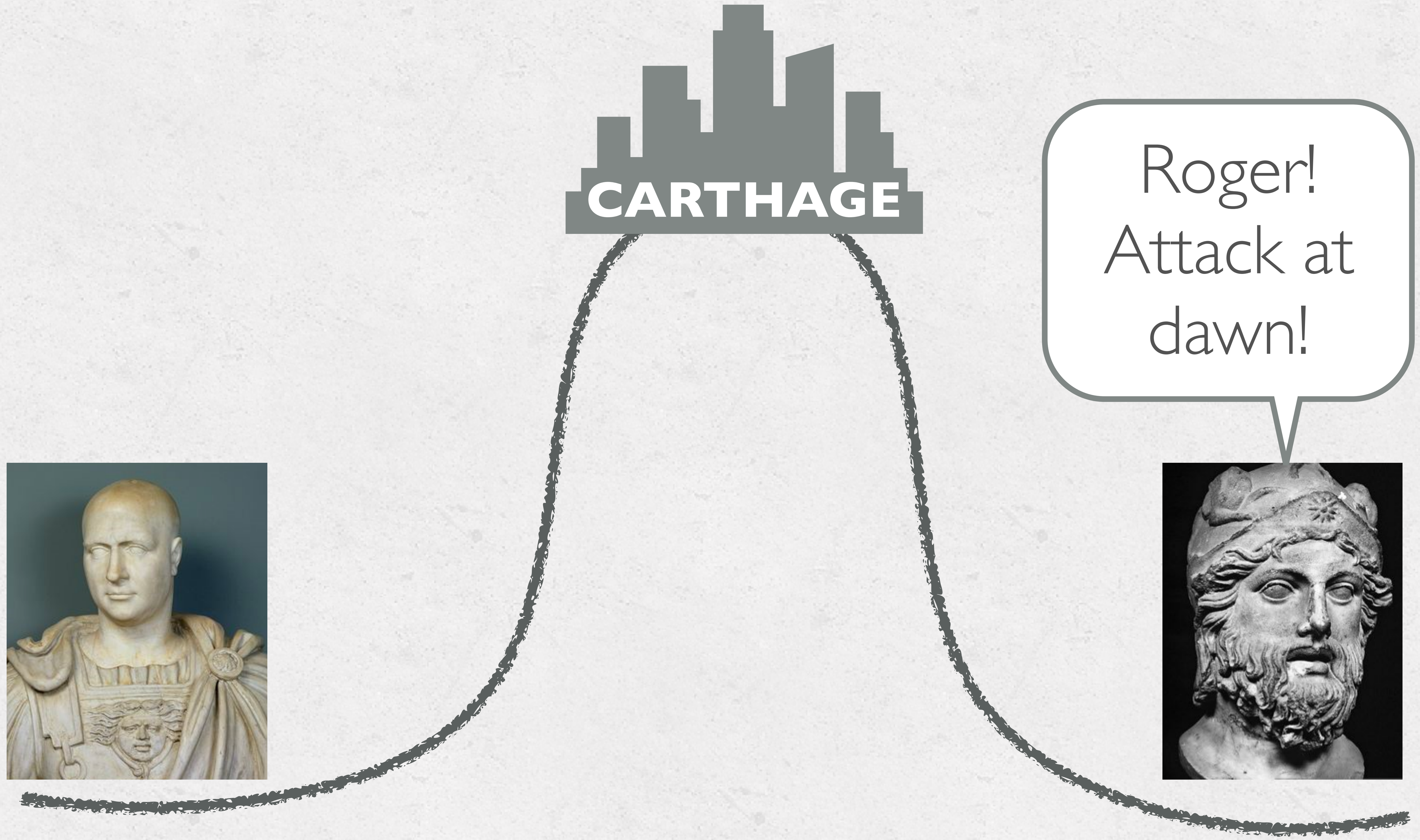
# Coordinated Attack

- Two generals, on opposite sides of a city on a hill.
- If they attack simultaneously, they will be victorious. If one attacks without the other, they will be repelled, routed.
- Can communicate by messenger. Messengers can get lost or be captured.
- How do they ensure they can take the city?



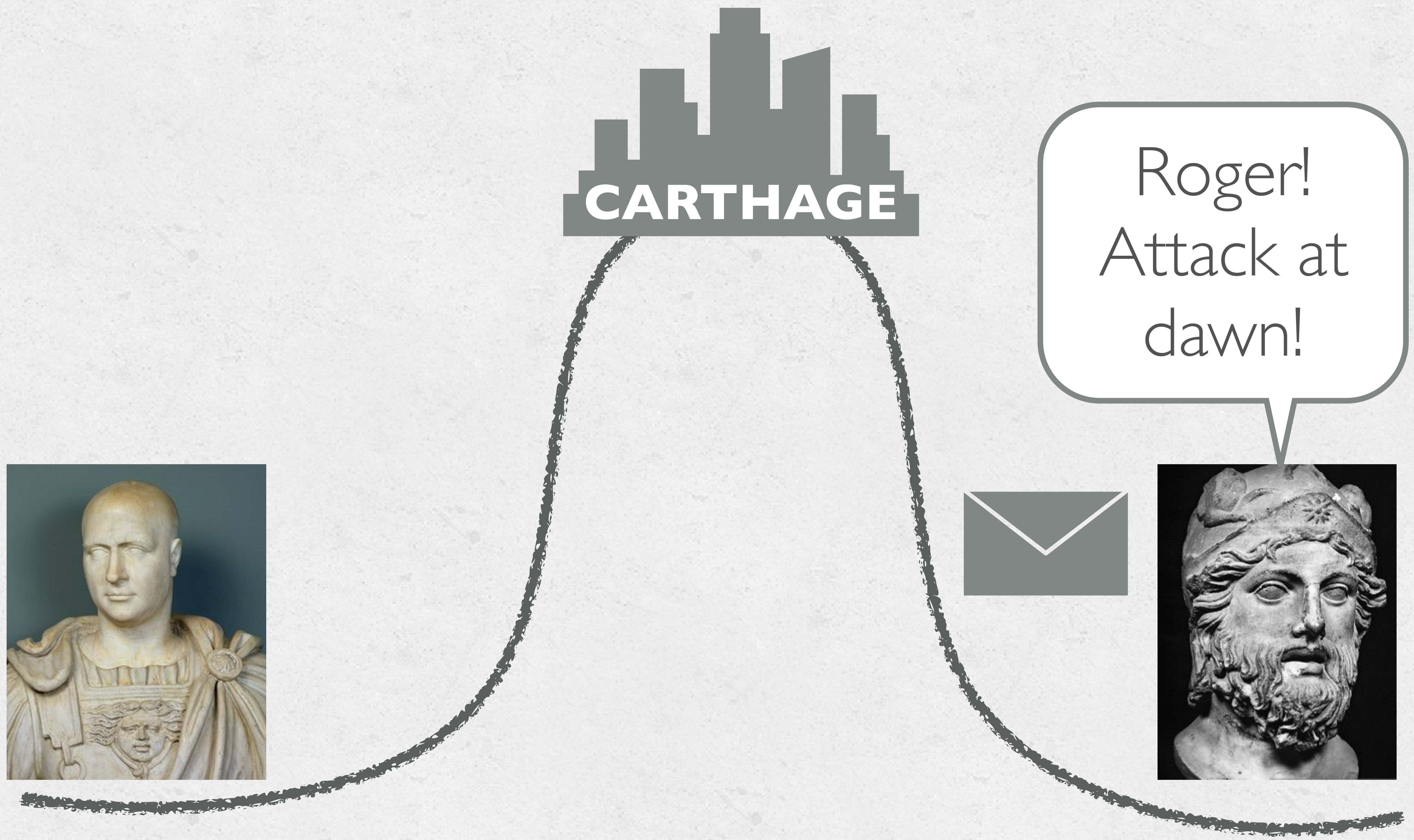
# COORDINATED ATTACK

- Two generals, on opposite sides of a city on a hill.
- If they attack simultaneously, they will be victorious. If one attacks without the other, they will be repelled, routed.
- Can communicate by messenger. Messengers can get lost or be captured.
- How do they ensure they can take the city?



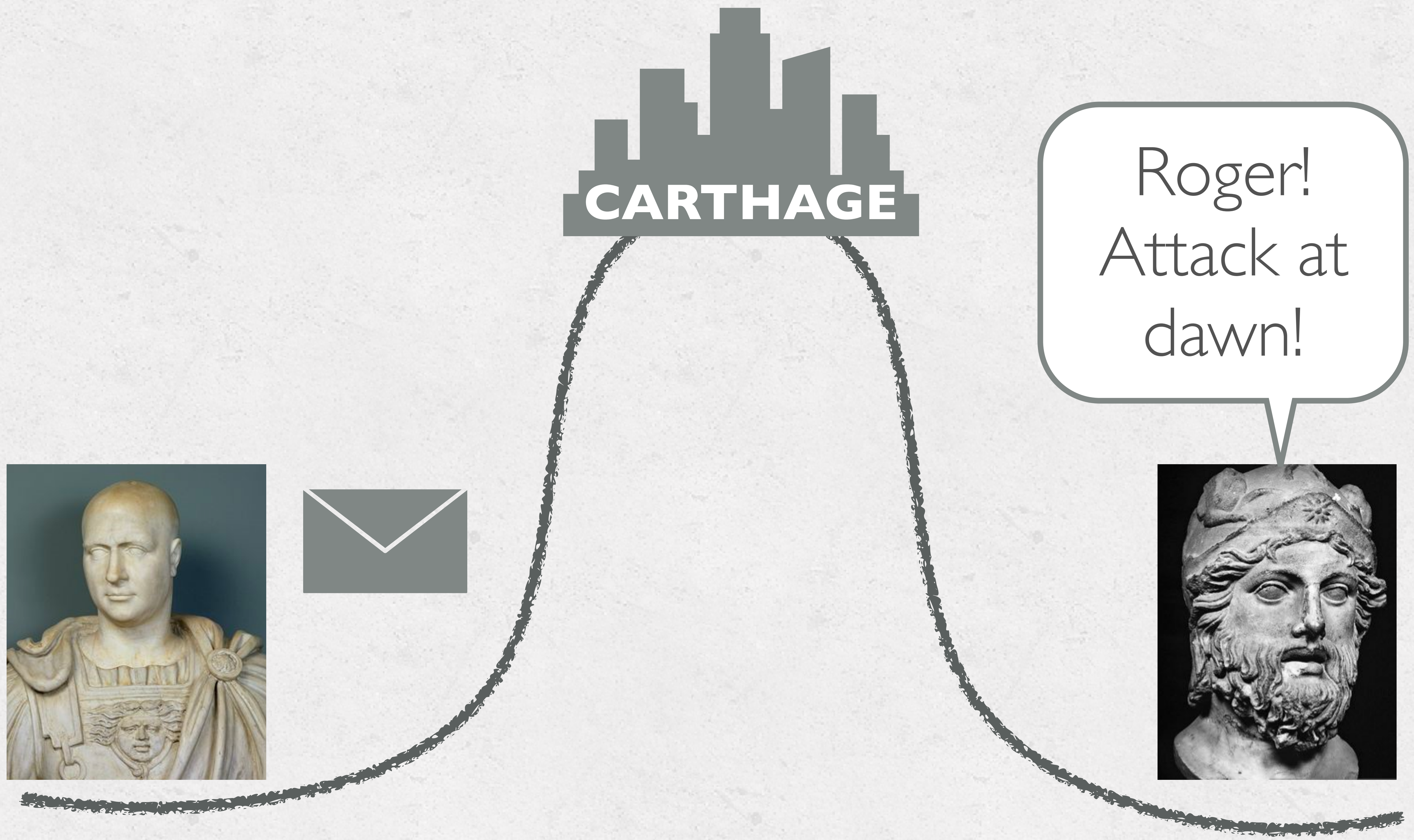
# Coordinated Attack

- Two generals, on opposite sides of a city on a hill.
- If they attack simultaneously, they will be victorious. If one attacks without the other, they will be repelled, routed.
- Can communicate by messenger. Messengers can get lost or be captured.
- How do they ensure they can take the city?



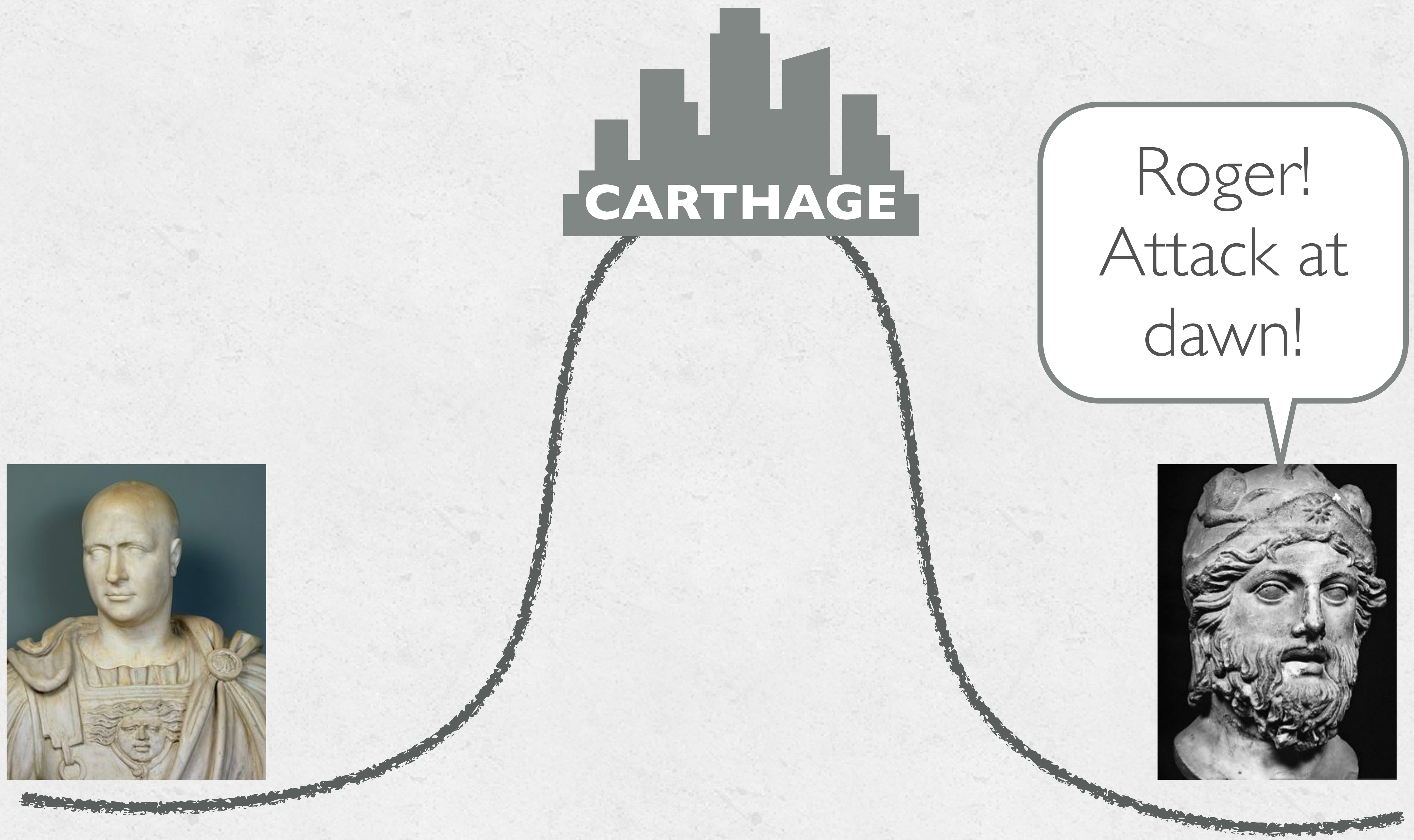
# Coordinated Attack

- Two generals, on opposite sides of a city on a hill.
- If they attack simultaneously, they will be victorious. If one attacks without the other, they will be repelled, routed.
- Can communicate by messenger. Messengers can get lost or be captured.
- How do they ensure they can take the city?



# Coordinated Attack

- Two generals, on opposite sides of a city on a hill.
- If they attack simultaneously, they will be victorious. If one attacks without the other, they will be repelled, routed.
- Can communicate by messenger. Messengers can get lost or be captured.
- How do they ensure they can take the city?



# Coordinated Attack

**Answer:** There does not exist a protocol to decide when and whether to attack.

**Proof by contradiction.** Assume a protocol exists. Let the minimum number of messages received in any terminating execution be $n$. Consider the last message received in one such execution.

The sender's decision to attack does not depend on whether or not the message is received; sender must attack. Since the sender attacks, the receiver must also attack when the message is not received.

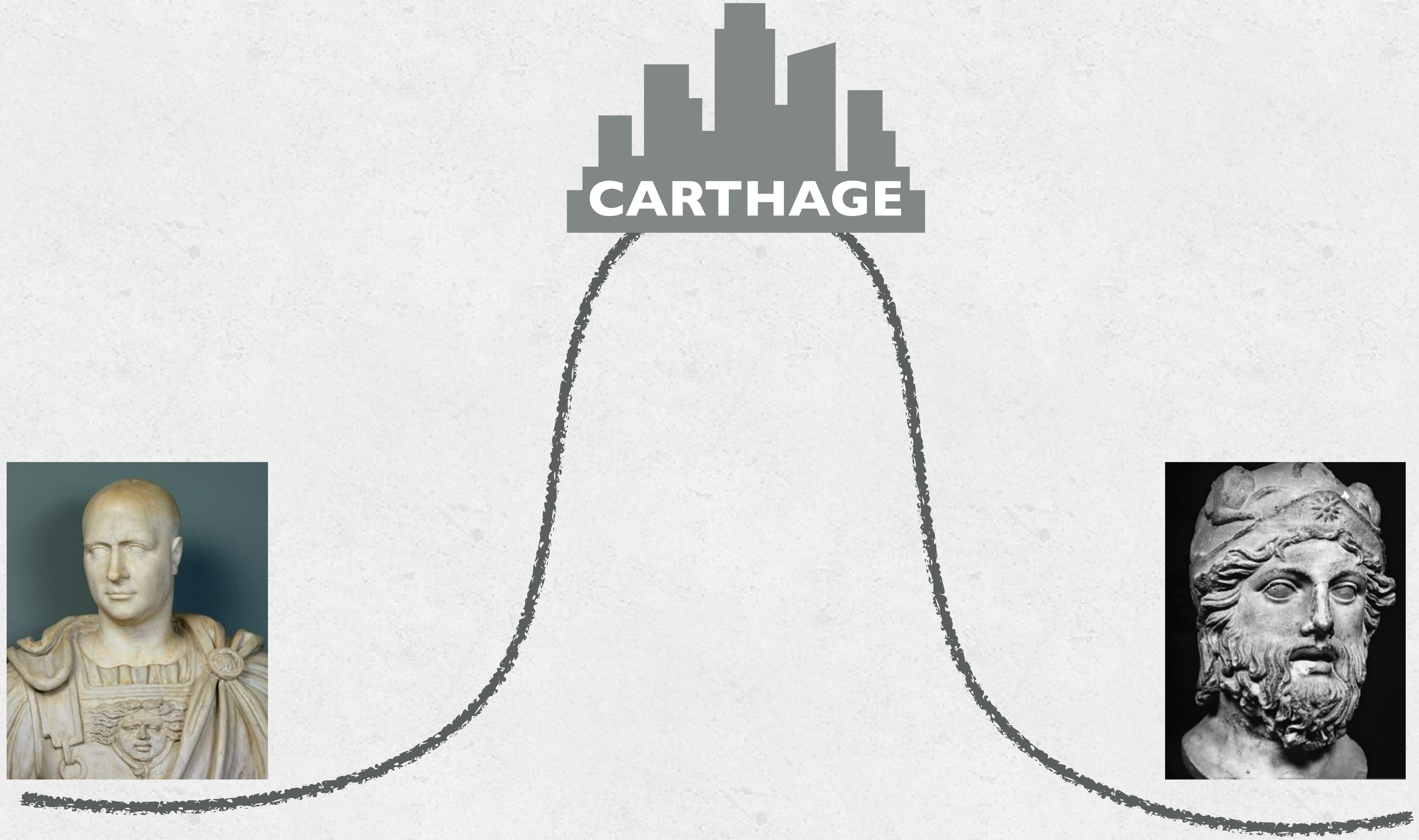
Therefore, the last message is irrelevant, and there exists an execution with $n$-1 message deliveries. $n$ was the minimum! Contradiction.

# Coordinated Attack

The coordinated attack problem requires *common knowledge!*

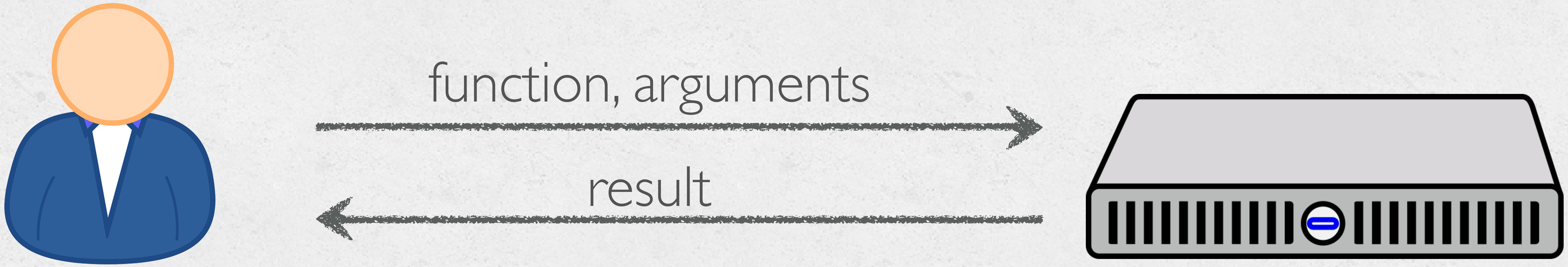
Each message only moves one step up in the hierarchy (i.e., $S\varphi \rightarrow E^1\varphi \rightarrow E^2\varphi \rightarrow E^3\varphi \rightarrow \ldots$),

never reaches $C\varphi$.



# Lab 1: Remote Procedure Calls

***Goal:*** *Execute function call on remote machine as if both the callee (server) and caller (client) were on the same machine.*




h/t for RPC slides: Tom Anderson

# Distributed Computing Model

- Processes (also called nodes/machines/servers) communicate by passing messages over a network
- Failure assumptions:
  - **No failures**
  - Fail-stop
  - Crashes
  - Byzantine
  - etc.
- Network assumptions:
  - Synchronous
  - **Asynchronous**
  - etc.

# Asynchronous Model

- Messages can be:
  - delayed indefinitely
  - dropped (indistinguishable from delayed)
  - duplicated
  - re-ordered

- No bound on clock skew across processes.

Weak assumptions ⇒ robust results

# RPC Semantics

- **At least once (lab 1b)** = when the call returns successfully on the client side, was executed at least once (maybe multiple times) on the server. Continuously retries until successful.

- **At most once** = if the call returns successfully, was executed once. Never executed more than once.

- **Exactly once (lab 1c)** = at most once + continuous retries

# Getting to At-Least-Once

- RPC library waits for response for a while
- If none arrives, re-send the request
- Do this until successful

# A Key/Value Store Example

- Client sends Put(k, v)
- Server receives it, responds with PutOk(), gets dropped by the network
- Client sends Put(k, v) again

What if the operation is an Append?

# But What About TCP?

- TCP: reliable bi-directional byte stream between two endpoints
  - Retransmission of lost packets
  - Duplicate detection
- But what if TCP times out and client reconnects?

# When At-Least-Once is Sufficient

When there are no side-effects and operations are *idempotent*.

Example: read-only operations.

# At-Most-Once

- Client includes unique ID (UID) with each request (same UID for re-send).
- Server RPC code detects duplicate requests, returns previous reply instead of re-running the handler.

```
if seen[uid] {
    r = old[uid]
} else {
    r = handler()
    old[uid] = r
    seen[uid] = true
}
```

# Some At-Most-Once Issues

- How do we ensure UID is unique?
  - Big random number?
  - Combine unique client ID (IP address?) with sequence number?
  - What if client crashes and restarts? Can it reuse the same UID?
  - In labs, nodes never restart. Equivalent to: every node gets new ID on start

# More Issues: When Can Server Discard Results?

- **Option 1:** Never?

- **Option 2:** unique client IDs, per-client RPC sequence numbers, client includes "seen all replies <= X" with every RPC

- **Option 3:** only allow client one outstanding RPC at a time, arrival of X+1 allows server to discard all <= X

Labs use Option 3 .

# Even More Issues

- Marshalling and parsing of messages?
- Version mismatch between client and servers?
- What if clients and servers crash and restart?

All out of scope for lab 1.

# Did We Win?

Well, no. Not yet (but we will...)

- What if the server does crash? And its disk fails?
- If we replicate, how do we ensure that replicas all execute the same RPCs (the same `Commands`) in the same order? ***Isn't that just the coordinated attack problem?***