



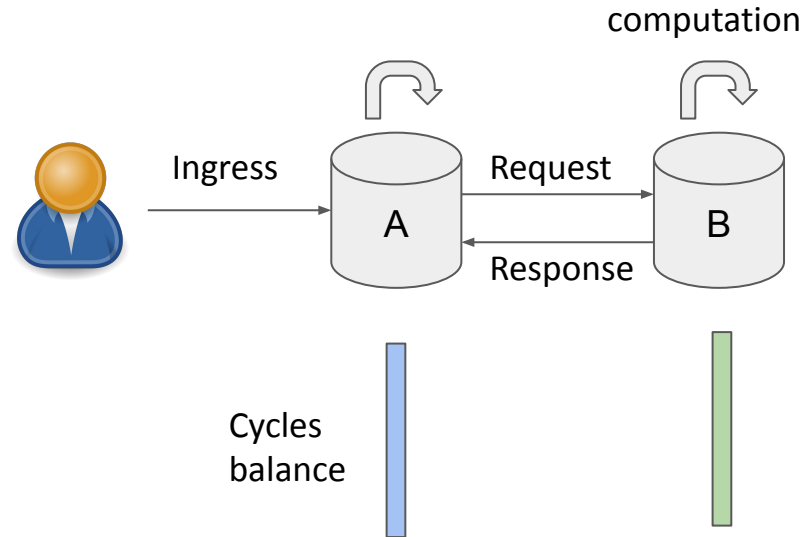
DFINITY

# **Charging cycles for running Canister smart contracts on the Internet Computer**

*Alexandra & Dimitris*

# Background

- Cycles act as the computational resource to execute actions on the Internet Computer
- ICP can be converted into Cycles (or get some from the Cycles faucet)
- 1T Cycles = 1 SDR



# Agenda

- Goals of charging cycles
- Design principles
- How charging works for various actions
- Special topics
  - “Freezing” canisters
  - Deep dive on message execution

# Goals of charging cycles

---

- Canisters pay for the resources used
- Certain DDoS attacks become harder

# Design principles

---

- “Reverse gas” model
  - End user pays (e.g. Ethereum) vs canister (smart contract) pays on the IC
- Two parts in cost function:
  - Fixed cost for basic operation
  - Variable based on “strain” put on the system
- Reserve upfront and refund excess

*Disclaimer: In the following we'll explain the current state of affairs wrt charging cycles. Things will likely change as the IC evolves, including both what actions are being charged for and their price.*

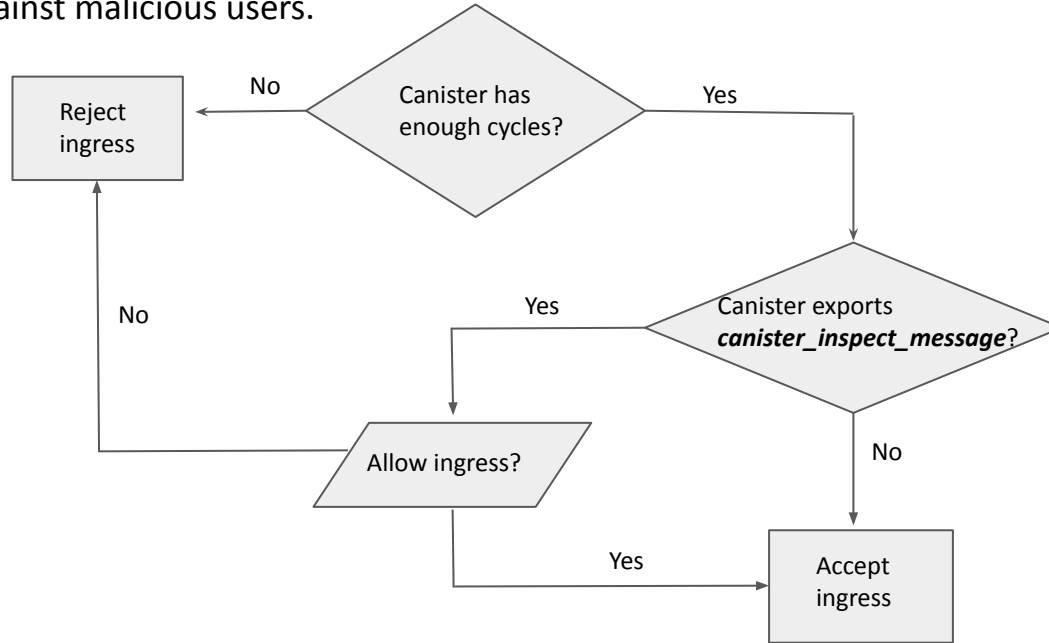
# Ingress messages

---

- Paying for an ingress covers the expense of doing consensus on the message.
- The receiving canister always pays for the ingress message.
- The induction cost of an ingress into the input queue of a canister consists of:
  - Fixed fee for receiving the ingress
  - Variable fee proportional to bytes transmitter
    - user-controlled payload(method name, payload, nonce)

# Ingress messages

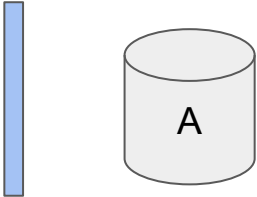
- A canister can filter and reject ingress messages through the ***canister\_inspect\_message*** method.
- Offers protection against malicious users.





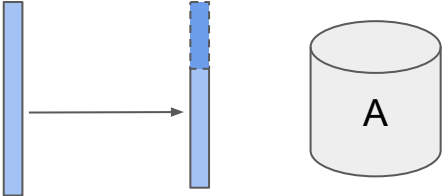
# Executing a message

- Fixed cost + cost proportional to Wasm instructions executed




# Executing a message

- Reserve max execution cost before execution start

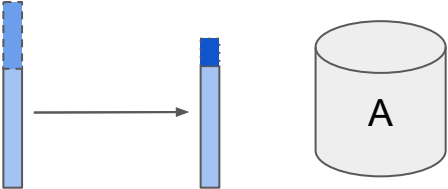





Cycle balance 

Reservation 

# Executing a message

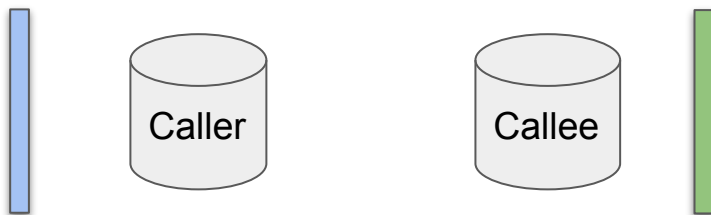
- Refund any excess once execution completes



Cycle balance   
Reservation   
Refund 

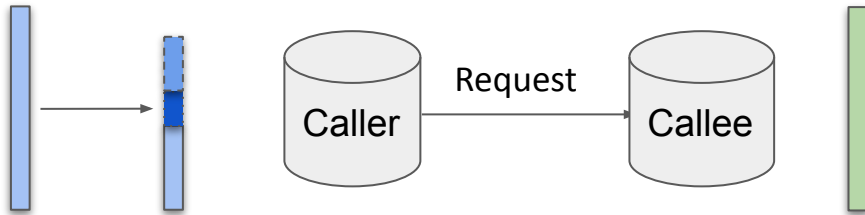
# Inter-canister calls



- **Transmission cost** of both request and response paid by caller
  - Fixed cost -> metadata of message
  - Proportional to bytes transmitted -> user controlled payload (method name + payload)
- **Execution cost** paid by executing canister
  - Caller pays for handling response
  - Callee pays for handling request





# Inter-canister calls

- Caller has produced a request for Callee.
- Caller's balance is updated:
  - Request transmission cost is deducted
  - Max response transmission cost is reserved
  - Max response execution cost is reserved



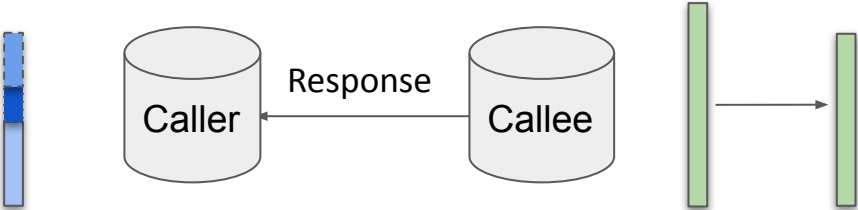
Cycle balance  

Response transmission reservation 

Response execution reservation 

# Inter-canister calls

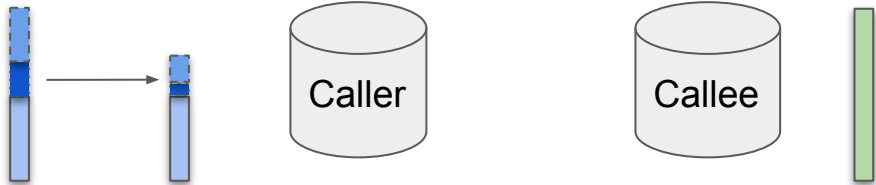
- Callee has processed Caller's request and has produced a response
- Response execution cost is deducted from Callee's balance



Cycle balance    
Response transmission reservation   
Response execution reservation

# Inter-canister calls

- Caller has processed Callee's response
- Response excess transmission cost is refunded
- Response excess execution cost is refunded



Cycle balance



Response transmission refund



Response execution refund



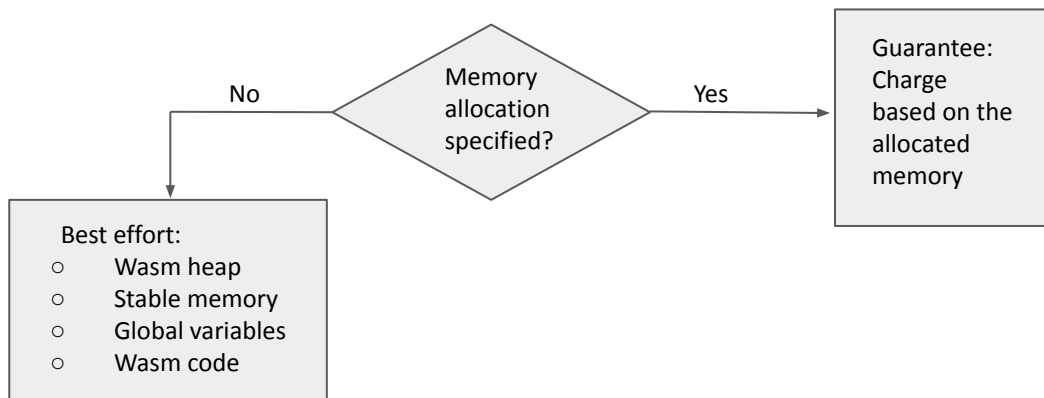
# Management messages

- If caller is an end user -> target (i.e. “managed”) canister pays
- If caller is a canister -> caller pays transmission cost
- Target canister pays for local execution
  - `install_code`
- Special case: `create_canister`
  - Can **only** be called by canisters
  - Incurs an additional flat cycles cost



# Resource allocation

- Compute allocation / scheduling priority - how often a canister is scheduled for execution
- Optional memory allocation setting
- Passive charges paid over time
- Canisters pay for these allocation even when they are idle



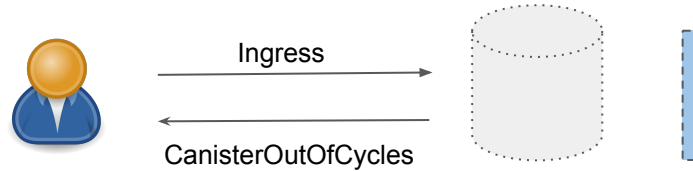
# “Freezing” canisters


---

- Freezing time setting.
- Preserves the canister’s code and state.
- Gives someone a chance to top-up the canister’s balance before it is uninstalled.
- The freezing threshold guarantees that the canister can pay for the resources during the allocated time.
  - cycles reserved for storage costs
  - cycles reserved for compute allocation

# “Freezing” canisters

- Induction of ingress messages may not be possible.  
freezing threshold + induction cost > canister's balance



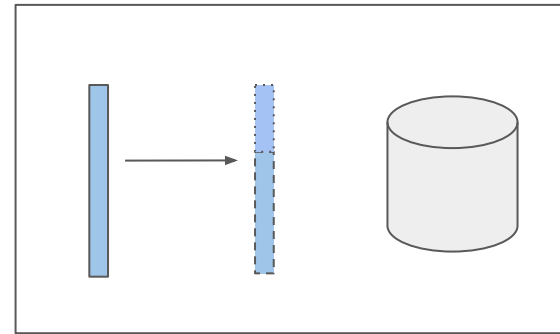
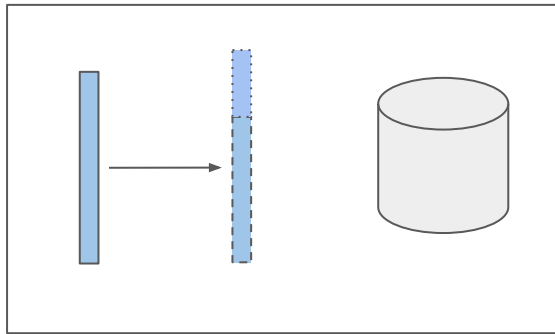
Ingress cost 




Freezing threshold 

# “Freezing” canisters

- A frozen canister cannot perform calls.

freezing threshold + message cost > canister's balance

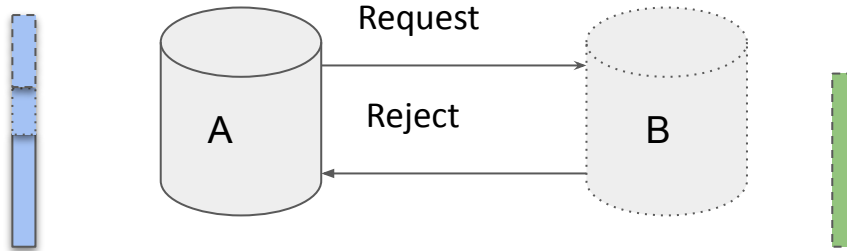





Cycle balance   
Message cost   
Freezing threshold 



# “Freezing” canisters

- A frozen canister will reject requests.

freezing threshold + execution cost > canister's balance

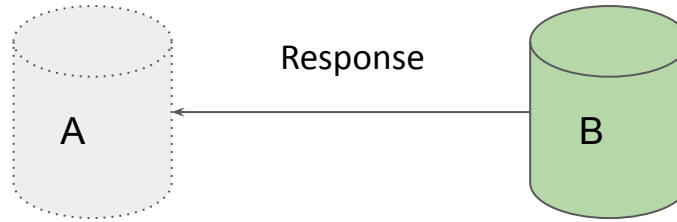
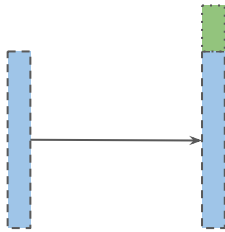


Cycle balance   
Transmission reservation   
Response execution reservation 

Execution cost   
Threshold freezing 

# “Freezing” canisters

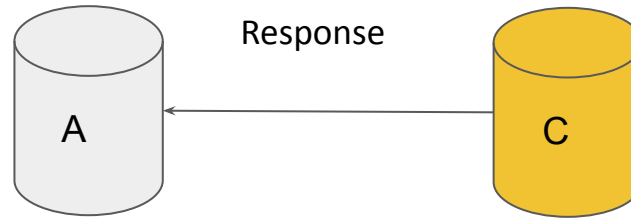
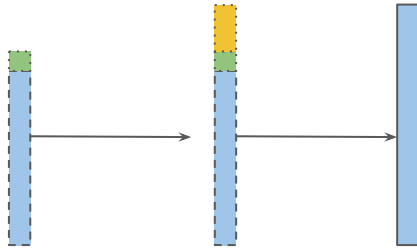
- A frozen canister can still process responses.
- Refunds can unfreeze a canister.







Refund   
Freezing threshold 

# “Freezing” canisters

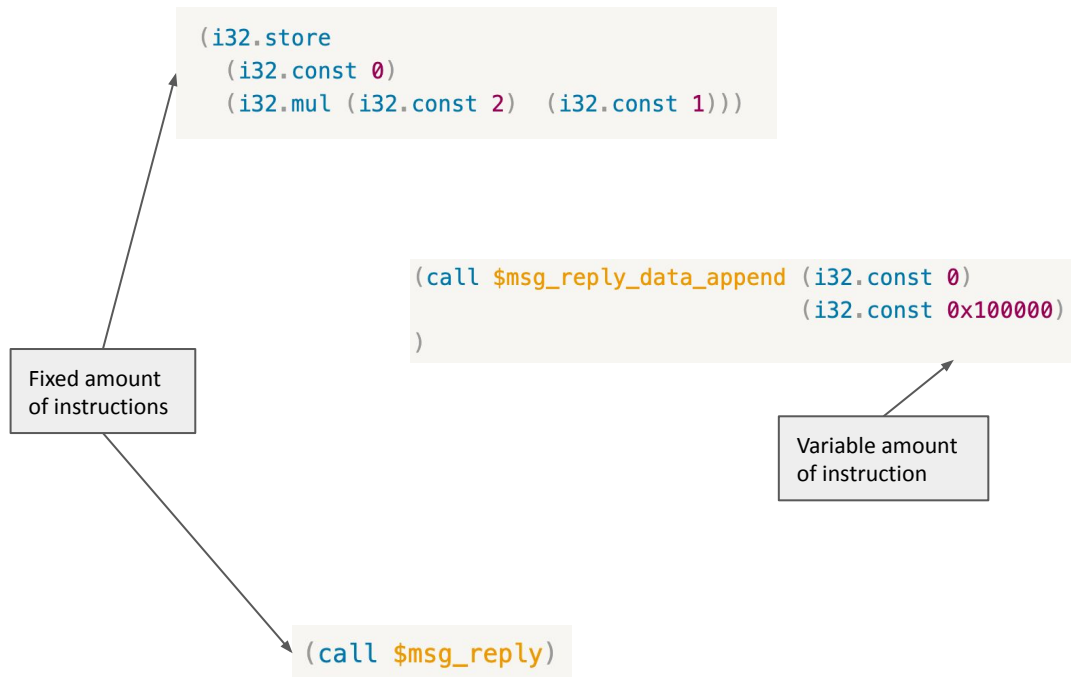
- A frozen canister can still process responses.
- Refunds can unfreeze a canister.



Refund    
Frozen balance   
Unfrozen balance 

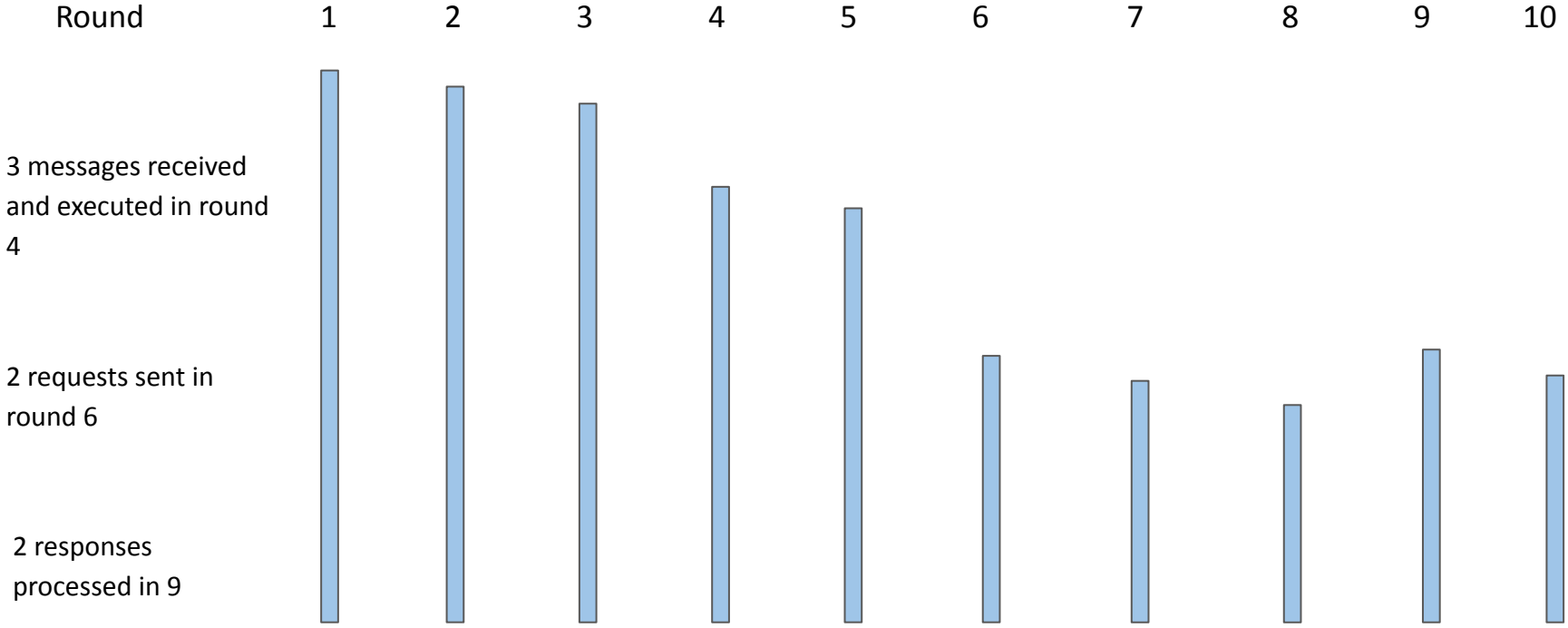
# Deep dive in message execution

- Counting the number of instruction executed
  - Operations with fixed amount of instructions
  - Operations with variable amount of instructions
- Extra overhead involved when copying data from one location to another.





# Lifecycle of a canister's Cycles balance



# Q & A

---



# Cost of running a dapp on the IC

- Current costs in <https://sdk.dfinity.org/docs/developers-guide/computation-and-storage-costs.html>
- Total cost heavily depends on the workload/storage requirements
- Working example:
  - 1000 ingress messages/sec, average size 100 bytes, average instructions consumption 100M
    - **Ingress cost:**  $1000 * (\text{ingress\_message\_reception\_fee} + 100 * \text{ingress\_byte\_reception\_fee}) = 1.4\text{B cycles/s}$
    - **Execution cost:**  $1000 * (\text{update\_message\_execution\_fee} + 10\text{M} * \text{ten\_update\_instructions\_execution\_fee}) = 40\text{B cycles/s}$
  - 10 xnet messages/sec, average size 200 bytes
    - $10 * (\text{xnet\_call\_fee} + 200 * \text{xnet\_byte\_transmission\_fee}) = 4.6\text{M cycles/s}$
  - 2GB of storage
    - $2 * \text{gib\_storage\_per\_second\_fee} = 254\text{K cycles/s}$