In [34]:
```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier

from sklearn.metrics import (
    accuracy_score,
    roc_auc_score,
    confusion_matrix,
    RocCurveDisplay,
    classification_report
)
from sklearn.utils.validation import check_is_fitted

import matplotlib.pyplot as plt
pd.set_option('display.max_columns', None)
np.set_printoptions(suppress=True)
```

In [35]:
```
dataset_path = "LabAssig5_stuff/churn.csv"

try:
    df = pd.read_csv(dataset_path)
    print(f"Dataset loaded successfully from '{dataset_path}'.")
except FileNotFoundError:
    print(f"Error: '{dataset_path}' not found.")

# Normalize
df["Churn"] = df["Churn"].astype(str).str.strip()
y = df["Churn"].map({
    "True": 1, "False": 0,
    "1": 1, "0": 0
}).astype(int)

X = df.drop(columns=["Churn"])
print("\nShape:", df.shape)
print("\nFirst 3 rows of the dataset:")
print(X.head(3))
```

```
Dataset loaded successfully from 'LabAssig5_stuff/churn.csv'.

Shape: (667, 20)

First 3 rows of the dataset:
  State  Account length  Area code International plan Voice mail plan  \
0    LA             117        408                 No              No
1    IN              65        415                 No              No
2    NY             161        415                 No              No

   Number vmail messages  Total day minutes  Total day calls  \
0                      0              184.5               97
1                      0              129.1              137
2                      0              332.9               67

   Total day charge  Total eve minutes  Total eve calls  Total eve charge  \
0             31.37              351.6               80             29.89
1             21.95              228.5               83             19.42
2             56.59              317.8               97             27.01

   Total night minutes  Total night calls  Total night charge  \
0                215.8                 90                9.71
1                208.8                111                9.40
2                160.6                128                7.23

   Total intl minutes  Total intl calls  Total intl charge  \
0                 8.7                 4               2.35
1                12.7                 6               3.43
2                 5.4                 9               1.46

   Customer service calls
0                       1
1                       4
2                       4
```

In [36]:
```python
# Identify numeric and categorical columns
num_cols = X.select_dtypes(include=np.number).columns.tolist()
cat_cols = X.select_dtypes(exclude=np.number).columns.tolist()


num_transformer = StandardScaler()
cat_transformer = OneHotEncoder(handle_unknown="ignore", sparse_output=False)

preprocessor = ColumnTransformer(
    transformers=[
        ("num", num_transformer, num_cols) if num_cols else ("num", "drop", []),
        ("cat", cat_transformer, cat_cols) if cat_cols else ("cat", "drop", []),
    ],
    remainder='passthrough'
)

print(f"\n# numeric features: {len(num_cols)}, # categorical features: {len(cat_cols)}")
print("Preprocessing pipeline created successfully.")
```

```
# numeric features: 16, # categorical features: 3
Preprocessing pipeline created successfully.
```

In [37]:
```python
strata = y if y.nunique() > 1 else None
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=strata
)

print("Train shape:", X_train.shape, " Test shape:", X_test.shape)
print("Target balance (train):", y_train.mean().round(4))
print("Target balance (test):", y_test.mean().round(4))
```

```
Train shape: (533, 19)  Test shape: (134, 19)
Target balance (train): 0.1426
Target balance (test): 0.1418
```

In [38]:
```python
# Bagging Classifier Pipeline
bagging_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("model", BaggingClassifier(
        estimator=DecisionTreeClassifier(random_state=42, max_depth=10),
        n_estimators=50,
        random_state=42,
        n_jobs=-1
    ))
])

# Random Forest Pipeline
random_forest_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("model", RandomForestClassifier(
        n_estimators=100,
        random_state=42,
        max_depth=10,
        n_jobs=-1
    ))
])

# Fit
print("Training Bagging Classifier...")
bagging_pipe.fit(X_train, y_train)
check_is_fitted(bagging_pipe)
print("Training complete.")

print("\nTraining Random Forest Classifier...")
random_forest_pipe.fit(X_train, y_train)
check_is_fitted(random_forest_pipe)
print("Training complete.")
```

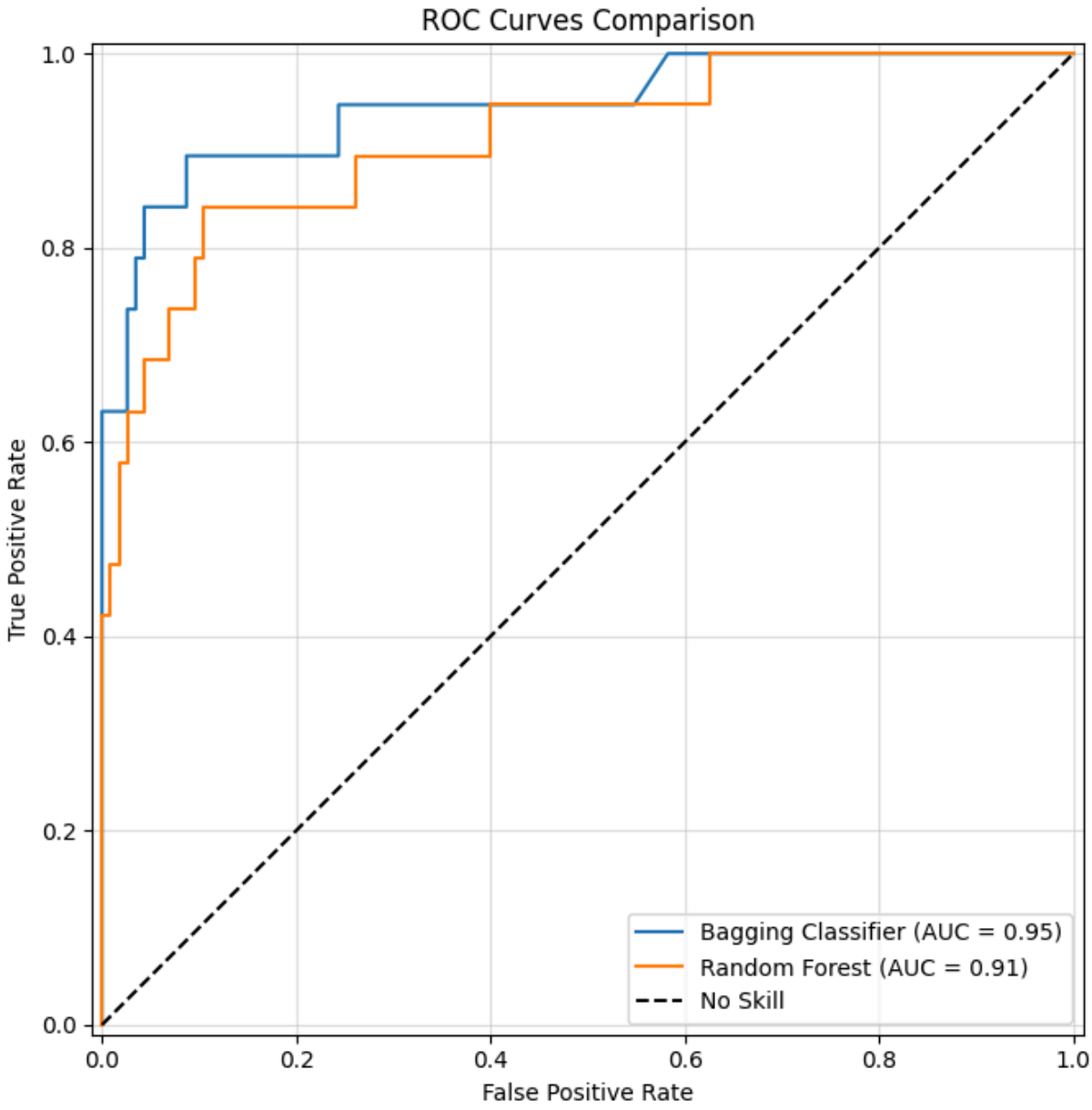
```
Training Bagging Classifier...
Training complete.

Training Random Forest Classifier...
Training complete.
```

In [39]:
```python
def evaluate_model(name, pipe, X_tr, y_tr, X_te, y_te, threshold=0.5):
    """Evaluates a classification model pipeline."""
    # probabilities
    y_tr_prob = pipe.predict_proba(X_tr)[:, 1]
    y_te_prob = pipe.predict_proba(X_te)[:, 1]

    # threshold
    y_tr_pred = (y_tr_prob >= threshold).astype(int)
    y_te_pred = (y_te_prob >= threshold).astype(int)

    # metrics
    acc_tr = accuracy_score(y_tr, y_tr_pred)
    acc_te = accuracy_score(y_te, y_te_pred)
    auc_tr = roc_auc_score(y_tr, y_tr_prob) if y_tr.nunique() > 1 else np.nan
    auc_te = roc_auc_score(y_te, y_te_prob) if y_te.nunique() > 1 else np.nan
    cm = confusion_matrix(y_te, y_te_pred, labels=[0, 1])
    clf_rep = classification_report(y_te, y_te_pred, labels=[0, 1], target_names=["No Churn", "Ch

    return {
        "model": name,
        "accuracy_train": acc_tr,
        "accuracy_test": acc_te,
        "roc_auc_train": auc_tr,
        "roc_auc_test": auc_te,
        "confusion_matrix_test": cm,
        "classification_report_test": clf_rep,
    }
```

```
print("Evaluation function defined.")
```

```
Evaluation function defined.
```

In [40]:
```
plt.figure(figsize=(8, 7))
ax = plt.gca()

try:
    RocCurveDisplay.from_estimator(bagging_pipe, X_test, y_test, name="Bagging Classifier", ax=ax
    RocCurveDisplay.from_estimator(random_forest_pipe, X_test, y_test, name="Random Forest", ax=a
    ax.plot([0, 1], [0, 1], "k--", linewidth=1.5, label="No Skill")
    plt.title("ROC Curves Comparison")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.grid(True, alpha=0.4)
    plt.legend()
    plt.tight_layout()
    plt.show()
except Exception as e:
    print("ROC plot could not be generated:", e)
```



In [41]:
```
results = [
    evaluate_model("Bagging Classifier", bagging_pipe, X_train, y_train, X_test, y_test),
    evaluate_model("Random Forest", random_forest_pipe, X_train, y_train, X_test, y_test),
]

# Create a summary metrics table
metrics_table = pd.DataFrame([{
    "Model": r["model"],
```

```
    "Accuracy (Train)": f"{r['accuracy_train']:.4f}",
    "Accuracy (Test)": f"{r['accuracy_test']:.4f}",
    "ROC AUC (Train)": f"{r['roc_auc_train']:.4f}",
    "ROC AUC (Test)": f"{r['roc_auc_test']:.4f}",
} for r in results]).sort_values(
    by=["ROC AUC (Test)", "Accuracy (Test)"], ascending=[False, False]
).reset_index(drop=True)

print("--- Model Performance Summary ---")
print(metrics_table)

# Print confusion matrices and classification reports
for r in results:
    print("\n" + "="*80)
    print(r["model"])
    print("\nConfusion Matrix (Test):")
    print(r["confusion_matrix_test"])
    print("\nClassification Report (Test):")
    print(r["classification_report_test"])
```

```
--- Model Performance Summary ---
                Model Accuracy (Train) Accuracy (Test) ROC AUC (Train)  \
0  Bagging Classifier           0.9944          0.9478          0.9999
1       Random Forest           0.9775          0.9104          0.9999

  ROC AUC (Test)
0         0.9460
1         0.9121

================================================================================
Bagging Classifier

Confusion Matrix (Test):
[[115   0]
 [  7  12]]

Classification Report (Test):
              precision    recall  f1-score   support

    No Churn       0.94      1.00      0.97       115
       Churn       1.00      0.63      0.77        19

    accuracy                           0.95       134
   macro avg       0.97      0.82      0.87       134
weighted avg       0.95      0.95      0.94       134


================================================================================
Random Forest

Confusion Matrix (Test):
[[114   1]
 [ 11   8]]

Classification Report (Test):
              precision    recall  f1-score   support

    No Churn       0.91      0.99      0.95       115
       Churn       0.89      0.42      0.57        19

    accuracy                           0.91       134
   macro avg       0.90      0.71      0.76       134
weighted avg       0.91      0.91      0.90       134
```

In [42]:

```
# best model
if metrics_table["ROC AUC (Test)"].notna().any():
    best_col = "ROC AUC (Test)"
else:
    best_col = "Accuracy (Test)"
```

```
best_idx = metrics_table[best_col].astype(float).idxmax()
best_name = metrics_table.iloc[best_idx]["Model"]
best_model_pipe = {"Bagging Classifier": bagging_pipe, "Random Forest": random_forest_pipe}[best_
print(f"Best model based on '{best_col}': {best_name}")
```

```
Best model based on 'ROC AUC (Test)': Bagging Classifier
```